Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Aljana Polanc

Aplikacija za pregled tehnologij spletnih projektov na podlagi avtomatske analize repozitorijev

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: doc. dr. Aleš Smrdel

Ljubljana, 2016

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

V okviru diplomskega dela razvijte aplikacijo, ki bo služila za vzdrževanje ažurne evidence o tehnologijah pri različnih spletnih projektih. Aplikacija mora omogočiti avtomatsko periodično pregledovanje vseh aktivnih projektov znotraj nekega podjetja in posodabljanje podatkov o uporabljenih tehnologijah za vsakega izmed projektov. V ta namen najprej analizirajte tehnologije, ki se uporabljajo pri izdelavi sodobnih spletnih projektov, nato pa realizirajte aplikacijo, ki se mora biti sposobna povezati z repozitorijem projektov, prebrati vse projekte in na podlagi analize datotek projektov razbrati tehnologije, ki so uporabljene pri posameznem projektu. Za vsak projekt mora aplikacija razbrati uporabljene programske jezike, orodja, knjižnice ter razvijalce, ki sodelujejo pri posameznem projektu. Razvijte tudi spletni vmesnik, ki bo omogočal prikaz podatkov o projektih in tehnologijah znotraj nekega podjetja ter nadroben prikaz tehnologij uporabljenih pri posameznem projektu. Razvijte tudi administrativni spletni vmesnik, ki bo omogočal dopolnjevanje in popravljanje avtomatsko pridobljenih podatkov. Pri razvoju aplikacije izberite najprimernejše tehnologije na strani odjemalca in na strani strežnika. Uspešnost delovanja razvite aplikacije testirajte na nekem repozitoriju projektov.

Zahvaljujem se mentorju doc. dr. Alešu Smrdelu za strokovno pomoč in odlično odzivnost. Hvala tudi vsem bližnjim, ki so me tekom študija spodbujali in mi stali ob strani.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
HTML	Hyper Text Markup Language	Hipertekstovni označevalni jezik
CSS	Cascading Style Sheets	Kaskadne stilske podloge
AJAX	Asynchronous JavaScript and XML	Asinhroni JavaScript in XML
DOM	Document Object Model	Dokumentni objektni model
API	Application Programming Interface	Aplikacijski programski vmesnik
MVC	Model-View-Controller	Model-pogled-nadzornik
URI	Uniform Resource Identifier	Unikaten identifikator vira
CRUD	Create, read, update and delete	Ustvari, beri, posodobi, izbriši
SQL	Structured Query Language	Strukturiran povpraševalni jezik
ORM	Object-relational mapping	Objektno-relacijsko preslikovanje
DSL	Domain-specific language	Domensko specifična nadgradnja jezika
JSON	JavaScript Object Notation	Notacija objektov JavaScript
YAML	YAML Ain't Markup Language	Ne-označevalni jezik za strukturiranje podatkov
SASS	Syntactically Awesome Stylesheets	Sintaktično odlične stilske podloge
HTTP	HyperText Transfer Protocol	Hipertekstovni prenosni protokol
REST	Representational State Transfer	Prenos predstavitvenega stanja

Povzetek

Naslov: Aplikacija za pregled tehnologij spletnih projektov na podlagi avtomatske analize repozitorijev

Diplomsko delo zajema razvoj in predstavitev aplikacije, katere namen je olajšati pregled nad tehničnimi in drugimi podatki spletnih projektov, ki se v podjetju razvijajo oziroma vzdržujejo. Aplikacija avtomatsko posodablja zbirko relevantnih podatkov spletnih projektov preko integracije s spletno storitvijo GitHuba, kjer iz izvorne kode projekta pridobi informacije o programskih jezikih, knjižnicah, drugih tehnologijah, sodelavcih na projektu in ostalih pomembnih podatkih. V prvem delu diplomske naloge smo se posvetili pregledu tehnologij in metod, ki se uporabljajo pri razvoju sodobnih spletnih aplikacij. Nato smo v drugem delu podrobneje predstavili potek razvoja aplikacije. Pri razvoju smo uporabili ogrodje Ruby on Rails, podatkovno bazo PostgreSQL, podatke pa smo procesirali z uporabo ogrodja Sidekiq. Aplikacijo smo preizkusili na testni množici odprtokodnih projektov ter njeno delovanje predstavili skozi zaslonske slike. V zaključku pa smo podali predloge za nadaljnji razvoj.

Ključne besede: spletna aplikacija, Ruby on Rails, GitHub, repozitorij.

Abstract

Title: Application for the technology overview of web projects based on automatic analysis of repositories

This thesis comprises the development and presentation of an application which aims to facilitate the overview of the technical and other data of web projects that are being developed or maintained in a company. The application automatically updates the collection of relevant data of the web projects by integrating with the GitHub web service, where it obtains the information regarding programming languages, libraries, other technologies, project contributors and other important data. In the first part of the thesis we focused on a brief overview of technologies and methods used when developing modern web applications. Then in the second part we described in detail the development of the application. To develop the application we used Ruby on Rails framework and PostgreSQL database, while we used Sidekiq framework to process the data. We tested the application on a set of various open source projects and presented its functioning using the screen shots. In conclusion we described proposals for further development.

Keywords: web application, Ruby on Rails, GitHub, repository.

Poglavje 1

Uvod

Več kot deset let že mineva, odkar so na konferenci Web 2.0 prvič definirali splet kot platformo za gradnjo aplikacij [1]. Vedno več namiznih aplikacij se seli v tako imenovani oblak, spletni brskalnik pa vse bolj nadomešča vlogo operacijskega sistema. Posledično je tudi podjetij, ki se ukvarjajo s spletnim razvojem, na pretek.

Od rojstva spleta pa do danes se je način spletnega razvoja drastično spremenil. Včasih je razvijalec spletno stran postavil tako, da je odprl urejevalnik besedila in pričel pisati preprosto označevalno, kasneje pa tudi skriptno kodo. Danes je spletni razvoj povsem drugačna zgodba, ki zajema množico različnih programskih jezikov, podatkovnih baz, ogrodij in knjižnic. Uveljavile so se številne dobre razvijalske prakse, kot so verzioniranje, avtomatizirano testiranje in zvezna integracija.

V poplavi vseh različnih tehnologij in orodij pa lahko podjetje, ki se ukvarja s spletnim razvojem, hitro izgubi pregled nad tem, katere tehnologije uporablja na obstoječih projektih. Tako se je porodila ideja o aplikaciji, ki bi omogočila evidenco vseh spletnih projektov, ki se v podjetju razvijajo, ter njihovih tehničnih podatkov. Ključna prednost aplikacije bi bila v tem, da se podatki posodabljajo avtomatsko, s čimer se izognemu nezaželenemu ročnemu vzdrževanju, ki bi najverjetneje povzročilo neažurnost podatkov.

Za razvoj aplikacije bomo uporabili priljubljeno ogrodje Ruby on Rails.

Aplikacijo bomo integrirali s spletno storitvijo GitHuba, kjer podjetja zelo pogosto gostujejo repozitorije projektov. Z analizo izvorne kode repozitorija bomo pridobili informacije o programskih jezikih, knjižnicah in drugih tehnologijah na posameznem projektu. Te informacije bomo nato predstavili skozi uporabniški vmesnik, ki bo ponujal učinkovit pregled nad tehničnimi podatki projektov. Menimo, da bi takšna aplikacija podjetjem lahko prinesla dodano vrednost, saj bi z njeno uporabo lahko izboljšali pretok informacij med razvijalci ter olajšali odločanje o izbiri primernih tehnologij za posamezni projekt.

Diplomsko nalogo smo razdelili na več delov. V prvem delu bomo nekoliko podrobneje predstavili problematiko, ki jo rešujemo. V drugem delu se bomo posvetili teoretičnemu pregledu metod in tehnologij, ki jih srečamo pri razvoju sodobnih spletnih aplikacij. Poznavanje področja je ključnega pomena za razvoj naše aplikacije. Gre za zelo obsežno področje, tako da se bomo bolj osredotočili na tiste dele, ki imajo večjo vlogo pri razvoju naše aplikacije.

Tretji, najobsežnejši del, zajema opis razvoja aplikacije. Začeli bomo z analizo zahtev, ki jim moramo zadostiti, tako z vidika uporabnika kot tudi z vidika notranje zasnove aplikacije. Nato bomo predstavili uporabljene tehnologije in podatkovni model, podrobneje pa bomo opisali arhitekturo in delovanje aplikacije. Na koncu bomo skozi zaslonske slike predstavili funkcionalnosti uporabniškega vmesnika.

V zaključku bomo pregledali rezultat opravljenega dela in podali predloge za nadaljnji razvoj.

Poglavje 2

Predstavitev problematike

Podjetja, ki se ukvarjajo z razvojem programskih rešitev za različne naročnike, istočasno razvijajo ali vzdržujejo mnogo projektov. Na trgu obstajajo številni produkti, ki olajšajo projektno vodenje, kot npr. Atlassian Jira in Basecamp, vendar pa nobeno od teh orodij ne daje dobrega vpogleda v tehnično plat projekta, ki se razvija. To pomeni, da ni enostavnega vpogleda v informacije, kot so programski jeziki in tehnologije, ki so na posameznem projektu uporabljeni.

Takšne informacije se lahko beležijo v raznih dokumentnih sistemih, vendar pa se tu pojavlja velik problem z vzdrževanjem dokumentacije, saj ta velikokrat postane zastarela, ker se je ne popravlja sproti, vzporedno z razvojem projekta, in ne odraža dejanskega stanja.

V primeru, da podjetje za gostovanje repozitorijev izvorne kode projektov uporablja GitHub, je pregled nad tehničnimi značilnostmi projektov do neke mere dostopen preko njihovega spletnega vmesnika, vendar pa so tu številne pomanjkljivosti. Razvijalci imajo po navadi dostop le do podatkov projektov, kjer tudi sodelujejo. Vpogled v tehnične informacije je neroden, saj ni vsega na enem mestu. Prav tako pa je do nekaterih podatkov mogoče priti le s pregledom izvorne kode.

Zaradi zgoraj naštetih pomanjkljivosti smo prišli na zamisel, da bi razvili aplikacijo, ki bo podatke, pridobljene na GitHubu, organizirala v primernejšo

obliko, predstavljeno v skladu s potrebami problema, ki ga rešujemo.

Glavni cilj aplikacije je, da izboljša pretočnost informacij o projektih in njihovih tehničnih značilnostih med razvijalci znotraj podjetja. Naslednji primeri ponazarjajo probleme, ki bi jih z uporabo aplikacije lahko odpravili.

- Imamo večje podjetje, ki je razdeljeno na razvojne skupine, ki delajo na posameznih projektih. Razvijalci znotraj skupine nimajo dobrih informacij o projektih in tehnologijah, ki jih uporabljajo v drugih skupinah oziroma vedo ravno toliko, kot lahko sproti izvedo preko pogovorov. Tak način širjenja informacij pa je zelo nesistematičen in neorganiziran. **Rešitev**: aplikacija bi vsakemu razvijalcu omogočila vpogled v osnovne podatke o vsakem projektu, ki se v podjetju razvija.

- Razvijalec bi npr. želel vedeti, ali je na kakšnem drugem projektu že uporabljena knjižnica, ki bi jo potreboval na projektu, na katerem trenutno dela, zato da lahko tam pogleda, kako je bila uporabljena oziroma, da ve, na koga se lahko obrne za pomoč. **Rešitev**: z aplikacijo lahko razvijalec poišče vse projekte, ki uporabljajo določeno knjižnico.

- Razvijalec je dobil nalogo, da na projektu vzpostavi avtomatsko testiranje. Na voljo je precej orodji, vendar je smiselno uporabiti tisto, ki se uporablja že na drugih projektih, in s katerim imajo tudi drugi razvijalci že izkušnje. Da bi pridobil te informacije, bi moral povprašati druge razvijalce in se zanesti na točnost njihovih odgovorov. Verjetno ne bi dobil odgovorov vseh oziroma bi moral na to informacijo čakati. **Rešitev**: z aplikacijo lahko pogleda, katera orodja se uporabljajo na obstoječih projektih.

Ideja je, da bi razvili spletno aplikacijo, ki bi podjetjem omogočila učinkovitejši pristop k omenjenim problemom. Aplikacija, ki jo želimo razviti, mora temeljiti na avtomatskem prepoznavanju tehničnih karakteristik spletnega projekta, kot so programski jezik, orodja, tehnologije in knjižnice. Koristno bi bilo tudi prepoznavanje razvijalcev, ki sodelujejo pri projektu. Za

izvedbo tega je ključno, da ima naša aplikacija dostop do izvorne kode oziroma repozitorijev projektov.

Primer: imamo spletno aplikacijo, ki je spisana v jeziku PHP in ogrodju Symfony2, za podatkovno bazo uporablja MySQL, pri predpomnjenju določenih podatkov pa je uporabljena nerelacijska baza Redis. Kot vmesnik za dostop do baze MySQL se uporablja knjižnica Doctrine ORM. Poleg tega aplikacija uporablja številne druge knjižnice, kot npr. Twig za predloge, Swift Mailer za pošiljanje elektronskih sporočil itd. Funkcionalnost aplikacije je pokrita z avtomatskimi testi, kar je izvedeno z uporabo knjižnice Behat. Poskrbljeno je tudi za zvezno integracijo, pri tem se uporablja oblačna storitev Travis CI. Naša spletna aplikacija bi v tem primeru razbrala naslednje karakteristike:

Jezik: PHP.

Ogrodje: Symfony2.

Podatkovne baze: MySQL, Redis.

Knjižnjice: Doctrine, Twig, Swift Mailer itd.

Testiranje: Behat, PHPUnit.

Zvezna integracija: Travis CI.

V prihodnosti bi funkcionalnost nadgradili, tako da bo aplikacija zmožna razbrati še več podrobnosti ter pokriti čim večji nabor tehnologij.

Poglavje 3

Razvoj sodobnih spletnih aplikacij

V tem poglavju bomo naredili hiter pregled tehnologij, orodij in metod, ki jih srečujemo pri razvoju sodobnih spletnih aplikacij, s poudarkom na odprtokodnih tehnologijah. Gre za precej obsežno področje, zato se bomo osredotočili predvsem na bistvene značilnosti, ki jih moramo upoštevati pri tehnični izvedbi naše aplikacije. Za razumevanje zahtev in funkcionalnosti aplikacije, ki jo razvijamo, je ključnega pomena seznanjenost s tehnologijami in orodji, ki se uporabljajo pri razvoju spletnih aplikacij.

3.1 Programski jeziki

Na področju razvoja spletnih aplikacij so se uveljavili predvsem programski jeziki, ki omogočajo produktivnost, agilnost in hitro prototipiranje. To so visokonivojski dinamični jeziki [2].

Prednost visokonivojskih jezikov je v tem, da en sam ukaz lahko izvede več tisoč strojnih ukazov, medtem ko jih nizkonivojski jeziki, kot je C, izvedejo precej manj [3]. Omogočajo avtomatsko upravljanje s pomnilnikom, napredne podatkovne strukture, visok nabor operacij, intuitivni aplikacijski programski vmesnik (angl. Application Programming Interface, v nadaljevanju API) za manipulacijo s podatki in njihovim stanjem [4]. Če dodamo še to, da so berljivejši, saj so ekspresivnejši in bližje psevdo-kodi ter imajo tako

položnejšo učno krivuljo [5], je razumljivo, da so postali najbolj priljubljena izbira pri razvoju spletnih aplikacij.

Dinamični jeziki za razliko od statičnih ne zahtevajo vnaprej definiranih spremenljivk ter njihovih tipov, kar razvijalcu omogoči večjo svobodo in boljšo produktivnost, vsaj pri manjših aplikacijah.

V skupino visokonivojskih dinamičnih jezikov med drugim spadajo tudi PHP, Ruby in Python. Gre za priljubljene jezike [2], v katerih je napisanih mnogo zelo znanih spletnih aplikacij. V zadnjem času postaja vse bolj priljubljen tudi JavaScript, ki se je dolgo časa uporabljal le kot jezik na strani odjemalca, s pojavom tehnologije NodeJS pa se je začel uporabljati tudi kot strežniški jezik. Razlog je poleg hitrosti tudi večja učinkovitost, saj lahko en sam razvijalec zgradi strežniški in odjemalski del aplikacije v istem jeziku ter celo z uporabo iste kode.

Kljub temu pa v določenih primerih dinamični jeziki niso tako razširjeni. V tradicionalnih poslovno-informacijskih sistemih, kjer je bolj kot hitrost razvoja, pomembna varnost in stabilnost, so še vedno bolj uveljavljeni jeziki, kot so Java ali C#, saj veljajo za robustnejše [6]. Druga slabost dinamičnih jezikov je počasnost, ki je pri večini spletnih aplikacij sicer zanemarljiva. V primerih, ko je visoka hitrost pomembna, bodisi zaradi narave operacij ali pa zaradi velikega števila uporabnikov oziroma spletnih zahtevkov, so velikokrat kritični deli aplikacij prepisani v hitrejše jezike, kot so C, Scala ali Go.

3.2 Ogrodja in knjižnice

Ogrodja za razvoj spletnih aplikacij so se začela pojavljati kmalu po razvoju skriptnih jezikov in so prav tako nastala z namenom povečanja produktivnosti in skrajšanja časa potrebnega za lansiranje aplikacije na trg. Poleg skrajšanja časa razvoja in s tem stroškov, ogrodja podpirajo dobre razvijalske prakse in s tem povečajo kvaliteto ter zanesljivost razvitih aplikacij.

Ogrodje predstavlja osnovo, na kateri se aplikacijo gradi naprej [7]. Glavna razlika med ogrodjem in knjižnico je inverzija kontrole – celoten tok programa

narekuje ogrodje in ne uporabnik s klicem funkcij kot pri knjižnici. Ogrodja izboljšajo in poenostavijo različne vidike razvoja spletnih aplikacij, tako da omogočijo:

- večplastno strukturo aplikacije po arhitekturnem vzorcu model-pogled-nadzornik (angl. Model-View-Controller), v nadaljevanju MVC,
- širok nabor pogosto uporabljenih funkcionalnosti in orodij pri razvoju,
- poenostavljeno upravljanje s paketi oziroma knjižnicami.

V nadaljevanju si bomo pobližje pogledali vse tri vidike.

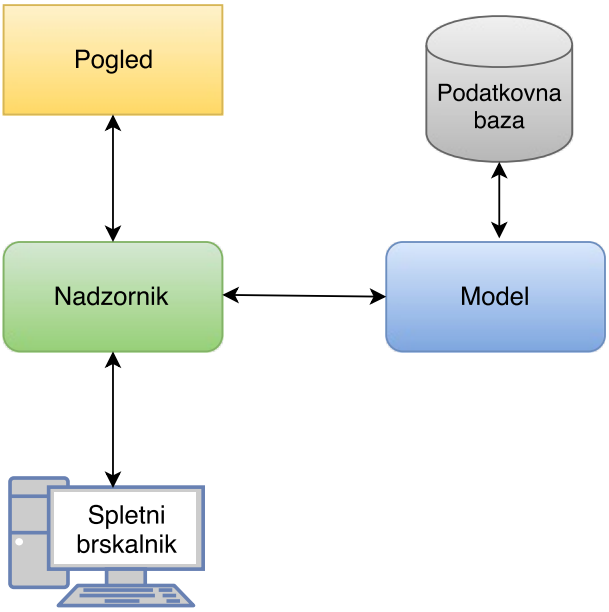
3.2.1 Arhitekturni vzorec MVC

Arhitekturni vzorec MVC je bil prvič predstavljen v programskem jeziku Smalltalk v 70-ih letih prejšnjega stoletja [8], in je torej starejši od spletnih aplikacij. Gre za standardno arhitekturo aplikacij z grafičnim uporabniškim vmesnikom, ki so jo posvojile tudi spletne aplikacije.

Aplikacije po navadi opravljajo različne naloge, zato se te združujejo v posamezne komponente, kar omogoča večji nadzor in lažje razumljivo kodo. Najpogostejša delitev je na tri plasti, in sicer podatkovno, logično in predstavitveno, kar velja tudi za vzorec MVC. Princip delovanja, razviden tudi iz slike 3.1, je sledeč:

- Model neposredno upravlja s podatki, logiko in pravili aplikacije. Po navadi je zgrajen iz razredov, ki komunicirajo s podatkovno bazo in implementirajo poslovno logiko. Zajema torej obnašanje aplikacije, neodvisno od uporabniškega vmesnika.
- Nadzornik (angl. Controller) komunicira z modelom, tako da mu posreduje vhodne podatke. Izhodne podatke pridobi iz modela ter jih posreduje naprej pogledu.

- Pogled (angl. View) skrbi za prikaz podatkov, tako da zgradi končni uporabniški vmesnik, kar v primeru spletnih aplikacij pomeni dokument HTML, ki ga potem prikaže spletni brskalnik.



Slika 3.1: Shema arhitekturnega vzorca MVC.

3.2.2 Funkcionalnosti in orodja

Podatkovna plast

Ogrodja vključujejo knjižnice, ki omogočajo enostaven dostop ter shranjevanje podatkov v bazi. Gre za tehniko objektno-relacijskega preslikovanja (angl. Object-relational mapping, v nadaljevanju ORM), ki skrbi za pretvarjanje podatkov iz podatkovne baze v objekte ter obratno. Poleg tega močno poenostavi pisanje poizvedb nad bazo, tako da jezik SQL nadomesti z intuitivnim in ekspresivnim programskim vmesnikom. Ogrodja pogosto vsebujejo tudi mehanizme za nadzor nad različicami baze in upravljanje z njenimi spremembami (t. i. migracije).

Abstrakcija protokola HTTP

Ena izmed ključnih komponent ogrodja je usmerjevalnik (angl. router), ki vsak zahtevek HTTP posreduje ustrezni metodi, ki je definirana v nadzorniku in izvaja določeno funkcionalnost. Razvijalec v konfiguracijski datoteki na preprost način definira pravila usmerjanja, tako da za posamezen URI ali glede na vzorec URI ter metodo HTTP (GET, POST, PUT, DELETE) definira preslikavo v metodo v nadzorniku, kjer se nahaja funkcionalnost. Poleg usmerjevalnika ogrodja omogočajo tudi lažji dostop in manipulacijo zahtevkov in odgovorov HTTP, npr. enostaven dostop do podatkov poslanih z metodo POST, upravljanje s piškotki in drugimi parametri v glavi zahtevka oz. odgovora.

Upravljanje s sejami in uporabniki

Implementacija uporabniškega sistema je običajno enostavna, saj ogrodja vsebujejo nabor ključnih komponent, kot so shranjevanje sej, kriptiranje gesla, sistem pravic in dostopa glede na različne uporabniške vloge. V nekaterih primerih pa so implementirane tudi že zaključene funkcionalnosti, kot so prijava in registracija uporabnika, posredovanje pozabljenega gesla ipd.

Varnost

Zaščita pred najpogostejšimi napadi na spletne aplikacije, kot so vrivanje SQL, XSS (angl. cross-site scripting) in CSRF (angl. cross-site request forgery) je v ogrodjih že vgrajena, tako da se razvijalcem s tem ni potrebno posebej ukvarjati.

Predloge

V predstavitveni plasti se uporablja poseben jezik za predloge, ki ima manjši nabor funkcionalnosti kot programski jezik ter dodatne funkcionalnosti za lažjo predstavitev podatkov. Obstaja mnogo knjižnic, ki implementirajo različne jezike za predloge, kot sta na primer Twig (PHP) in pa Handlebars.js

(JavaScript). Prednost uporabe predlog je ločevanje zadolžitev (angl. separation of concerns), kar povzroči boljšo preglednost in lažje vzdrževanje kode. Poleg tega razvijalcem ni potrebno poznati programskega jezika, da lahko kodirajo na predstavitveni plasti.

Orodja in generatorji

Sodobna ogrodja vsebujejo številna orodja, ki se izvajajo v ukazni vrstici in poenostavijo nadzor nad aplikacijo: upravljanje z bazo, predpomnilnikom, generiranje osnovne strukture ipd.

Razhroščevanje

Razhroščevanje je poenostavljeno, saj ogrodja zabeležijo napake, ki se v aplikaciji zgodijo in omogočajo podroben pregled nad težavo.

3.2.3 Upravljanje paketov in medsebojnih odvisnosti

Upravljalec paketov (angl. package manager) je orodje, ki razvijalcem olajša uporabo in nadzor nad programskimi knjižnicami ter njihovimi medsebojnimi odvisnostmi. Pred pojavom tovrstnih orodij, se je izvorno kodo knjižnic ročno skopiralo med datoteke projekta, kar pa je neučinkovito in težko za vzdrževanje.

Upravljalci paketov se med jeziki razlikujejo, v grobem pa so njihove zadolžitve podobne: preko omrežja naložijo paket s knjižnico, ki se nahaja v namenskih repozitorijih, skopirajo datoteke na pravilno mesto v projektu ter nato razrešijo tranzitivne odvisnosti [9].

Paketni upravljalec je sestavljen iz:

- repozitorija, kjer se nahajajo paketi s knjižnicami,
- odjemalca, ki komunicira z rezpozitorijem in izvaja prej omenjene naloge; običajno je to program v ukazni vrstici.

Za primer vzemimo jezik Ruby. Posamezen programski paket, ki vsebuje knjižnico, se imenuje Gem. RubyGems je orodje, ki zna namestiti pakete, repozitorij s paketi pa se nahaja na spletnem naslovu RubyGems.org. Za lažje upravljanje z njimi se v projektih uporablja program Bundler, ki zna namestiti pakete na podlagi seznama, ki je definiran v datoteki Gemfile, hkrati pa njihove različice in medsebojne odvisnosti zapiše v datoteko Gemfile.lock.

3.3 Podatkovne baze

Nepogrešljiv del spletne aplikacije je podatkovna baza. Večje ali bolj kompleksne spletne aplikacije uporabljajo tudi več različnih tipov podatkovnih baz, glede na potrebe. V osnovi ločimo relacijske (SQL) in nerelacijske (NoSQL) baze.

Relacijske baze temeljijo na tabelah in relacijah med tabelami, imajo točno definirano strukturo oziroma shemo. Med odprtokodnimi rešitvami sta najbolj priljubljeni MySQL in PostgreSQL, med plačljivimi pa Oracle [10].

V nerelacijske baze uvrščamo tiste, ki ne sledijo standardnemu relacijskemu modelu in ne uporabljajo jezika SQL za pisanje poizvedb. Podatke shranjujejo na različne načine, ki se v grobem delijo na štiri kategorije [11]:

- Ključ-vrednost (angl. key value storage) (npr. Redis, Memcached) - uporabne za shranjevanje informacij o seji, uporabniškem profilu ali preferencah.
- Stolpične baze (angl. column database) (npr. Cassandra, HBase) – zapisovanje števcev, agregacija logov, primerno, če je veliko operacij pisanja.
- Dokumentne baze (npr. MongoDB, CouchDB) - primerne za analitiko, bloge oziroma v primerih, kjer niso potrebne transakcije.
- Grafne baze (npr. Neo4j) - za podatke, ki jih lahko modeliramo v

obliki grafov ali mrež ter nato računamo najkrajšo pot ali sorodnost, npr. socialna omrežja in priporočilni sistemi.

3.4 Tehnologije na strani odjemalca

HTML, CSS in JavaScript spadajo med tehnologije odjemalca, kar pomeni, da jih interpretira oziroma izvaja spletni brskalnik uporabnika. V zadnjih letih se je to področje zelo razvilo, kar je pripeljalo do pojavitve bogatih internetnih aplikacij (angl. Rich Internet Applications). Spletne aplikacije so začele postajati vse bolj podobne namiznim oziroma so se namizne aplikacije začele seliti na splet, saj gre za vedno zmogljivejšo platformo. Tudi tehnologije na strani odjemalca so zato prevzele določene karakteristike strežniških tehnologij. Področje je še zmeraj v razvoju, zato obstaja ogromno različnih pristopov k razvoju, izobilje orodij in knjižnic ter žal tudi problemov s kompatibilnostmi med njimi.

3.4.1 HTML

HTML je označevalni jezik, ki se sestoji iz posameznih značk [12]. Značke definirajo strukturo spletnega dokumenta, tako iz semantičnega vidika kot tudi oblikovnega, saj določajo postavitev posameznih elementov in s tem tudi grobo obliko spletnega dokumenta. Najnovejša različica HTML5 definira številne nove značke za opis semantike dokumenta (npr. <nav>, <article>, <footer>), nudi izboljšano podporo multimediji (avdio, video, canvas), integracijo vektorske grafike SVG ter specificira nove API-je [13].

3.4.2 CSS

CSS so kaskadne stilske podloge, ki omogočajo definiranje natančnejšega izgleda posameznih elementov HTML preko raznih parametrov, kot so vrsta pisave, odmiki, barve itd [14]. Tehnologija se zelo hitro razvija in konstantno

prinaša nove funkcionalnosti, kot so npr. razni vizualni učinki in transformacije slik, animacije elementov ipd.

Obstajajo številne nadgradnje jezika CSS; t. i. predprocesorji, ki temeljijo na drugem skriptnem jeziku, ki se nato prevede v CSS. Med najbolj priljubljene spadajo SASS, LESS in Stylus [15]. Njihova prednost je predvsem nadgradnja jezika, ki osnovnemu jeziku doda nove funkcionalnosti, kot sta na primer uporaba spremenljivk in koncepti iz objektno usmerjenih jezikov (npr. dedovanje), kar omogoča večjo modularnost, organizacijo in recikliranje kode ter s tem tudi večjo produktivnost.

Večjo produktivnost prinaša tudi uporaba ogrodij za CSS. Najbolj znani ogrodji sta Bootstrap in Foundation [16]. Ogrodja vsebujejo različne komponente in funkcionalnosti, npr:

- poenotenje izgleda med posameznimi spletnimi brskalniki (datoteka reset.css),
- mrežo (angl. grid) kot osnovo k odzivnemu spletnemu oblikovanju,
- nabor ikon, gumbov, elementov za obrazce ali dele uporabniškega vmesnika (modalna okna, zavihki ipd).

3.4.3 JavaScript

JavaScript je visokonivojski skriptni jezik, ki statične spletne dokumente obogati z dinamičnimi elementi in omogoča interakcijo z uporabnikom [17].

Jezik je sicer objektno usmerjen, vendar ima neobičajen objektni model, ki temelji na prototipiranju. Za premostitev specifik jezika so se razvile številne alternative, kot so CoffeScript, TypeScript ipd.

CoffeScript je t. i. sintaktični bonbon [18], ki po vzoru jezikov Python in Ruby naredi JavaScript konsistentnejši in berljivejši. Ker spletni brskalniki tega jezika ne podpirajo, ga je potrebno pretvoriti nazaj v JavaScript.

TypeScript [19] je Microsoftova odprtokodna nadgradnja jezika JavaScript. Podpira tudi originalno sintakso jezika, ki pa jo razširja s tipičnimi

koncepti, poznanimi iz klasičnih objektno usmerjenih jezikov, kot so razredi, vmesniki, imenski prostori, generiki, anotacije ter statični tipi.

Poleg tega obstaja tudi standard ECMAScript [20], ki ga implementira JavaScript in nekateri drugi jeziki, npr. ActionScript (Flash). Spletni brskalniki trenutno v celoti podpirajo različico ECMAScript 5. Novejša različica ECMAScript 6 oz. krajše ES6 je precejšnja nadgradnja jezika in pokriva večino funkcionalnosti jezikov TypeScript in CoffeScript ter tudi mnoge druge. Težava je v tem, da trenutno še nima podpore vseh najbolj popularnih spletnih brskalnikov, vsaj ne v celoti [21]. Za premostitev te ovire se lahko uporabi prevajalnik Babel, ki kodo napisano v ES6 prevede v različico, kompatibilno z vsemi spletnimi brskalniki.

Ena izmed pomembnih uporab JavaScripta je tudi za asinhrone klice na strežnik, z uporabo skupka tehnologij AJAX [22], kar omogoči interakcijo s strežnikom brez ponovnega nalaganja celotne spletne strani ter tako prispeva k boljši uporabniški izkušnji.

Knjižnice in ogrodja

JavaScript se velikokrat uporablja skupaj s knjižnicami, kot so jQuery, ki olajša manipulacijo dokumentnega objektnega modela (angl. Document Object Model, v nadaljevanju DOM) ter izvajanje klicev AJAX, in pa underscore.js, ki obogati jezik z veliko funkcijskimi principi. Za večje ali enostranske (angl. single-page application) spletne aplikacije se uporabljajo ogrodja, ki večinoma temeljijo na variacijah arhitekture MVC. Med priljubljena ogrodja sodijo Backbone.js (v kombinaciji z Marionette), Ember.js in AngularJS.

Ogrodja se med seboj razlikujejo po funkcionalnostih in filozofijah, na katerih temeljijo. Backbone.js je npr. zelo majhno ogrodje, ki pušča razvijalcem precej svobode pri implementaciji, medtem ko je Ember.js celovitejšo ogrodje, ima velik nabor funkcionalnosti in orodij ter temelji na principu "dogovor pred nastavitvami" (angl. convention over configuration).

Temeljne značilnosti in funkcionalnosti ogrodij so:

- deklarativna povezava med podatki modela in pogleda (HTML); ko se lastnost modela spremeni, se avtomatsko osveži tudi pogled, ali pa se to zgodi preko dogodkov in poslušalcev teh dogodkov,
- podpora predlogam, kot je Handlebars,
- usmerjevalnik, ki posreduje med URI-ji in metodami ter dobra podpora RESTful aplikacijam,
- ločitev manipulacije DOM-a od poslovne logike aplikacije, kar izboljša strukturo aplikacij.

Upravljanje paketov in razvijalska orodja

Node.js (krajše Node) je izvajalno okolje, ki omogoča razvoj in izvajanje strežniških aplikacij v JavaScriptu. Mnogo spletnih aplikacij je v celoti zgrajenih v tehnologiji Node, velikokrat pa se Node uporablja le kot razvijalsko orodje.

Upravljanje s paketi izvaja upravljalec NPM (Node Package Manager) [23], vendar v tem primeru ne gre za knjižnice, ki se uporabljajo v spletnem brskalniku na strani odjemalca, temveč za strežniške knjižnice oziroma orodja, ki se uporabljajo med razvojem in se izvajajo v ukazni vrstici.

Primera takšnih orodji sta npr. Grunt in Gulp, ki omogočata definiranje in izvajanje opravil izgradnje (angl. build tasks) [24]. Opravila izgradnje se razlikujejo glede na izvajalno okolje. V razvojnem okolju velikokrat v ozadju teče program, ki spremlja spremembe na datotekah in na njih izvaja definirana opravila, kot npr. prevajanje datotek iz SASS v CSS ali iz CoffeScript v JavaScript. V produkcijskem okolju pa je potrebno optimizirati hitrost nalaganja spletne strani, kar pomeni izvajanje nalog združevanja datotek ter minifciranja in zakrivanja kode.

Za upravljanje s paketi na strani odjemalca se pogosto uporablja Bower. Namesto njega pa se lahko pakete upravlja tudi z uporabo knjižnic Browserify ali webpack. Delujeta tako, da knjižnice, ki jih razvijalec naloži preko upravljalca NPM, lahko uporabi tudi na strani odjemalca. Poleg tega omogočata modularno organizacijo kode [25].

3.5 Nadzor različic

Verzioniranje je postopek ohranjanja več različic izvorne kode aplikacije, kar omogoči lažji nadzor nad spremembami v kodi, razveljavitev sprememb ter vzporedni razvoj različnih funkcionalnosti [26].

Ločimo centralizirane sisteme za verzioniranje (npr. SVN) ter porazdeljene sisteme, kot so Git in Mercurial. Git je trenutno najbolj uveljavljen, zato si ga bomo v nadaljevanju podrobneje pogledali.

Git je porazdeljen sistem, kar pomeni, da ne obstaja centralni strežnik, temveč se operacije lahko izvajajo lokalno na vsaki delovni kopiji repozitorija, neodvisno od centralnega repozitorija ali omrežnega dostopa, kar omogoča zelo hitro delovanje [27].

Repozitorij je osrednja komponenta, kjer je shranjena izvorna koda programa. Nad repozitorijem lahko razvijalec izvaja različne operacije, med katerimi so najpogostejše prevzem izvorne kode repozitorija in kreiranje delovne kopije (*checkout*), posodabljanje delovne kopije s spremembami iz repozitorija (*fetch* oz. *pull*), objava lokalnih sprememb v oddaljen repozitorij (*commit* in *push*).

Repozitorij je sestavljen iz več razvojnih vej (angl. branch), kar omogoča vzporeden razvoj različnih funkcionalnosti. Po zaključku razvoja se te veje združujejo v skupno razvojno vejo (angl. develop). Ko je aplikacija pripravljena na objavo, pa se razvojna veja združi v glavno vejo (angl. master), kjer se nahaja produkcijska različica aplikacije. Takšen delovni tok omogoča učinkovito ločevanje razvojnih vej in s tem različic aplikacije glede na okolje, v katerem se izvaja.

GitHub je najbolj priljubljen ponudnik gostovanja repozitorijev Git. Medtem ko je *git* program, ki teče zgolj v ukazni vrstici, GitHub temelji na spletnem vmesniku [28], ki poleg gostovanja repozitorijev in standardnih operacij verzioniranja, omogoča tudi številne druge funkcionalnosti, ki olajšajo nadzor izvorne kode in razvojni proces nasploh. Med pomembnejše funkcionalnosti spadajo:

- dokumentacija obogatena z jezikom Markdown,
- upravljanje z zahtevki (angl. issue),
- zahteve za združevanje kode (angl. pull request), ki omogočajo pregled kode in komentiranje,
- statistike repozitorija (št. prispevkov sodelujočih ipd.),
- nastavitve pravic dostopa,
- integracija z drugimi aplikacijami.

3.6 Namestitev spletne aplikacije in konfiguracija infrastrukture

V zadnjem času se je razširil koncept DevOps (angl. Developer Operations). Gre za precej ohlapno definiran koncept, v osnovi pa bi lahko rekli, da gre za skupek strokovnjakov, procesov in orodij, ki predstavljajo podporo razvoju, predvsem pa izboljšanju kvalitete in časa dostave aplikacij. Obstajajo številna orodja, ki poenostavljajo in avtomatizirajo:

- postavitev razvojnih okolij,
- konfiguracijo in vzdrževanje strežnikov,
- namestitev spletne aplikacije na strežnik.

Programska oprema Vagrant omogoča enostavno postavitev razvojnega okolja v virtualnem okolju, kot je VirtualBox, ali pa v vsebovalnikih Docker, ki aplikacijo povsem izolirajo od gostujočega sistema in s tem omogočijo prenosljivost med sistemi, pa tudi večjo fleksibilnost ter boljše performanse [29].

V datoteki Vagrantfile, ki je napisana v jeziku Ruby, je določena osnovna konfiguracija sistema (npr. podatki o sistemskih virih in omrežnih nastavitvah), podrobneje pa jo določajo skripte, ki so lahko napisane v Bashu ali pa

v drugih sistemih za konfiguracijo, predstavljenih v nadaljevanju. Razvijalec lahko v nekaj minutah s preprostim ukazom *vagrant up* postavi razvojno okolje. Na ta način se zagotavlja prenosljivost in enostavnost postavitve okolja med razvijalci.

Med orodja za upravljanje konfiguracije spadajo Puppet, Chef, Ansible, in druga [30]. Lahko se jih uporabi za postavitev razvojnega okolja v kombinaciji z Vagrantom ali pa za upravljanje konfiguracije produkcijskih strežnikov, s čimer je mogoče zagotoviti enako razvojno in produkcijsko okolje. Ansible je med njimi najnovejši in velja za najenostavnejšega, Puppet je primernejši za tiste, ki nimajo močnega razvijalskega znanja, Chef pa je precej kompleksen in se dobro obnese, če upravljamo z ogromnim številom strežnikov.

Proces postavitve infrastrukture je tako obravnavan na enak način kot pisanje kode (angl. Infrastructure as code). Namesto pisanja skript v Bashu, se konfiguracija definira na enostavnejši način v domensko specifični nadgradnji jezika (v nadaljevanju DSL) ali v datotekah tipa YAML (človeku prijazen standard za serializacijo podatkov). Konfiguracija se potem vzdržuje v sistemu za verzioniranje, kar omogoča učinkovit pregled nad spremembami in razveljavitvami le-teh. Sisteme je tako lažje replicirati, vzdrževati in zagotavljati konsistenco med njimi.

Proces objave spletne aplikacije na produkcijski strežnik se razlikuje glede na kompleksnost aplikacije. Vsekakor je že pri enostavnih spletnih aplikacijah smiselno ta proces avtomatizirati. V ta namen se lahko uporabi orodje Capistrano [31], obstajajo pa tudi druge podobne rešitve, kot npr. Fabric (Python) in Mina (Ruby). Capistrano uporablja DSL jezika Ruby in je zelo lahko razširljiv. Omogoča objavo aplikacij, napisanih tudi v drugih jezikih. Obstaja veliko prilagoditev za različna ogrodja, npr. ogrodje Symfony2 uporablja različico, ki se imenuje Capifony. V osnovi deluje tako, da se preko protokola SSH poveže na oddaljene strežnike, ki so razdeljeni po skupinah na različne vloge ter glede na vlogo izvaja vnaprej definirane ukaze.

3.7 Zvezna integracija

Zvezna integracija je razvojna praksa, ki omogoča hitro in redno dostavo sprememb, kar predstavlja osnovo agilnemu razvoju programske opreme.

Praksa spodbuja čim bolj pogosto združevanje posameznih razvojnih vej v skupno razvojno vejo. Pred združevanjem se izvede izgradnja (angl. build), kjer se postavi okolje, v katerem teče aplikacija. Nato se izvedejo avtomatizirani testi, ki preverijo pravilnost novih funkcionalnosti ter kompatibilnost nove kode z obstoječimi funkcionalnostmi [32]. Temelj zvezne integracije in dostave tako predstavljajo avtomatizirani testi. V nadaljevanju je opisanih nekaj najpogostejših oblik.

Testi enot služijo testiranju najmanjših individualnih enot kode, npr. testiranje posamezne metode v razredu, neodvisno od delovanja morebitnih drugih metod. Le-te se zamenja s posebnimi konstrukti (angl. mock, stub), ki simulirajo njihovo delovanje, zato da se doseže izolacijo testirane metode.

Funkcijski testi delujejo na principu črne škatle, tako da testirajo pravilnost izhodnih podatkov glede na vhodne. Primer je npr. testiranje delovanja API-ja, tako da se preveri, če zahtevek z določenimi parametri vrne pravilno strukturiran odgovor.

Integracijski testi so namenjeni testiranju komponent in komunikaciji med njimi, tako s funkcionalnega kot tudi performančnega vidika.

Regresijski testi zagotavljajo pravilnost delovanja obstoječih funkcionalnosti po preoblikovanju oz. refaktoriranju kode. Eden izmed ciljev regresijskih testov je odkriti, če spremembe enega dela aplikacije vplivajo na druge dele aplikacije.

Namestitev in konfiguracija lastnega sistema za zvezno integracijo je precej kompleksna in nepotrebna za večino aplikacij. Obstaja kar nekaj odprtokodnih rešitev pa tudi ponudnikov gostovanih storitev zvezne integracije, ki se izvaja v oblaku (primer: Travis CI, CircleCI). Prednost le-teh je v enostavnosti postavitve, integraciji z gostovanimi repozitoriji (npr. GitHub), kar pa predstavlja tudi slabost, saj ponudnik tako dobi tudi dostop do izvorne kode aplikacije.

Pogosto govorimo tudi o zvezni dostavi, ki predstavlja logično nadaljevanje zvezne integracije. Deluje tako, da po vsaki uspešni integraciji sproži postopek avtomatske objave nove različice aplikacije v produkcijsko ali pa v t. i. staging okolje, ki je čim bolj podobno produkcijskemu in je namenjeno ročnemu testiranju aplikacije pred objavo [33].

Poglavje 4

Zahteve in funkcionalnosti aplikacije

4.1 Zamisel

Namen aplikacije je olajšati pregled nad projekti in tehnologijami, ki se v podjetju razvijajo, ter omogočiti boljšo pretočnost informacij in znanja med razvijalci. Ciljni uporabniki aplikacije so razvijalci, zaposleni v podjetjih, ki se ukvarjajo z razvojem spletnih strani in aplikacij.

Ključna funkcionalnost aplikacije in njena glavna prednost je, da se vsi podatki o projektih in uporabljenih tehnologijah pridobijo povsem avtomatsko, kar pomeni, da ročno vzdrževanje ni potrebno.

Zaradi tega je naša aplikacija zasnovana na predpostavki, da ima dostop do izvorne kode projektov, saj lahko s pomočjo branja ustreznih datotek dobi podatke o uporabljenih programskih jezikih in tehnologijah. Ker so spletne aplikacije v veliki večini verzionirane, lahko z dostopom do repozitorija izvorne kode pridobimo tudi točne informacije o razvijalcih ter njihovih prispevkih.

Kot osnovo za pridobivanje podatkov o projektih smo izbrali GitHub, saj je izredno priljubljen sistem za gostovanje repozitorijev tudi med podjetji in ima poleg tega tudi dobro dodelan API, ki nam omogoča lahek dostop do

vseh potrebnih informacij.

Aplikacija je zastavljena kot interni sistem, ki ga podjetje namesti na strežnik, kar pomeni, da ena kopija aplikacije upravlja le s podatki enega podjetja. Če bi v prihodnje želeli sistem zastaviti kot oblačno rešitev, bi morali poleg tega razviti še uporabniški sistem, ki bi ločeval podatke glede na podjetje.

Ideja je, da razvijemo aplikacijo po principu vitke metodologije, kar pomeni, da razvijemo minimalni nabor funkcionalnosti, ki pa so dovolj pomembne, da z njimi osmislimo uporabo aplikacije in prinesemo dovolj veliko dodano vrednost.

4.2 Funkcionalnosti

4.2.1 Pregled zahtev z vidika uporabnika

Nabor potrebnih funkcionalnosti zajema:

- Pregled vseh projektov z osnovnimi informacijami o vsakem projektu (ime, opis, uporabljeni programski jeziki in tehnologije).
- Profil posameznega projekta:
  - Osnovni podatki o projektu, kot sta npr. ime in opis.
  - Seznam razvijalcev, ki so prispevali k projektu ter število njihovih prispevkov (angl. commits).
  - Seznam programskih jezikov, ki so na projektu uporabljeni.
  - Seznam uporabljenih knjižnic.
  - Seznam drugih uporabljenih tehnologij.
- Iskalnik, ki uporabniku omogoča poiskati projekte ali knjižnice glede na iskani niz.
- Profil knjižnice, ki ima naslednje funkcionalnosti:

  - Prikaz osnovnih podatkov, kot so ime, opis, ključne besede in povezava na domačo stran.
  - Možnost pošiljanja komentarja in priporočila (oz. odsvetovanja) s strani uporabnika.
  - Pregled komentarjev in priporočil.
  - Pregled projektov, kjer je knjižnica uporabljena.

- Vstopna stran, ki prikazuje nekaj zanimivih statistik glede uporabljenih programskih jezikov in tehnologij.

- Administracijski vmesnik, kjer lahko skrbnik aplikacije vnese podatke za dostop do informacij o spletnih projektih, npr. dostop do repozitorijev na GitHubu, od koder se potem podatki o projektih prenesejo v aplikacijo.

4.2.2 Pregled zahtev z vidika notranje zasnove aplikacije

- Podpora projektom, ki repozitorij z izvorno kodo gostujejo na GitHubu.

  Na GitHub smo se omejili, ker je zelo priljubljen ponudnik gostovanja repozitorijev, tudi med podjetji, saj omogoča privatne repozitorije ter še mnogo drugih funkcionalnosti, ki so pomembne za podjetja.

- Razširljivost glede načina dostopanja do repozitorija projekta, kar pomeni podporo tudi alternativam GitHuba (npr. Bitbucket).

  Aplikacijo je torej potrebno zasnovati tako, da lahko sprejme podatke o projektih iz različnih virov ter pretvori podatke v enotno obliko. Glavni pomislek tu je vprašljivost zmogljivosti API-ja drugih ponudnikov, saj ni nujno, da podpirajo enake funkcionalnosti kot GitHub. V tem primeru bi se morali omejiti le na storitve, ki so skupne vsem ponudnikom.

- Podpora spletnim projektom, ki so razviti v ogrodju Symfony2 (PHP), Ruby on Rails in okolju NodeJS.

Ker ima vsak programski jezik oziroma ogrodje svojo zgradbo datotek in knjižnic, smo se omejili le na nekaj bolj priljubljenih.

- Razširljivost v smislu dodajanja podpore novim jezikom in tehnologijam.

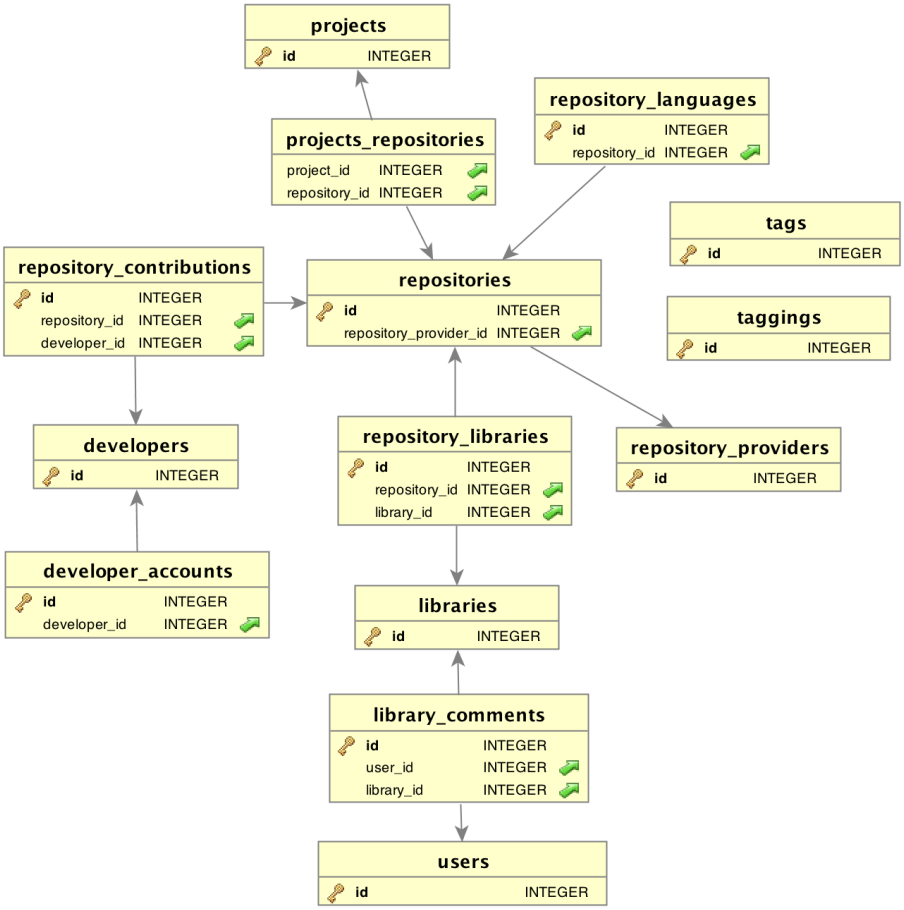
  Spletna aplikacija mora biti tako zasnovana, da je razširjanje podpore novim jezikom in zaznavanje novih tehnologij trivialno.

Poglavje 5

Podatkovni model

Aplikacije podatke, ki jih pridobi iz repozitorijev, shranjuje v podatkovno bazo. Glede na zahteve, opisane v prejšnjem poglavju, smo izdelali podatkovni model, predstavljen na sliki 5.1. V osnovi je podatkovni model zgrajen iz podatkov o:

- repozitorijih oziroma projektih,
- razvijalcih,
- programskih knjižnicah in
- tehnologijah.



Slika 5.1: Shema podatkovnega modela, prikazane so tabele s primarnimi in tujimi ključi. Za podroben opis posameznih tabel glej opise entitet v besedilu.

Repository (repozitorij)

Je osrednja entiteta, ki predstavlja posamezen repozitorij izvorne kode in vsebuje naslednje podatke:

- ime,
- programsko ime (angl. slug),
- opis,
- število prispevkov (angl. commit),
- primaren programski jezik,
- primarno ogrodje,
- status (aktiven oz. neaktiven).

RepositoryContribution (prispevki repozitorija)

Ta entiteta vsebuje podatke o številu prispevkov posameznega razvijalca v določen repozitorij, struktura je sledeča:

- referenca repozitorija,
- referenca razvijalca,
- št. prispevkov.

RepositoryLibrary (knjižnice repozitorija)

Ta entiteta pa povezuje repozitorije s knjižnicam ter vsebuje naslednje podatke:

- referenca repozitorija,
- referenca knjižnice,
- različica knjižnice.

RepositoryLanguage (jeziki repozitorijev)

Ta entiteta je namenjena hranjenju podatkov o programskih jezikih, ki so uporabljeni v posameznem repozitoriju, ter količini kode spisane v tem jeziku:

- referenca repozitorija,
- programski jezik,
- količina kode v bajtih.

Library (knjižnica)

Je entiteta, ki vsebuje podatke o posameznih programskih knjižnicah:

- ime,
- programsko ime (slug),
- opis,
- URL do repozitorija,
- URL do domače spletne strani,
- programski jezik.

Nekatere knjižnice vsebujejo tudi seznam ključnih besed, ki pa se shranjuje v splošno entiteto z oznakami.

LibraryComment (komentarji knjižnic)

Ta entiteta je nosilec podatkov o komentarjih in priporočilih vezanih na knjižnico. Atributi so naslednji:

- referenca na knjižnico,
- referenca na uporabnika,
- komentar,
- priporočilo (da/ne).

Developer (razvijalec)

Je entiteta, ki opisuje razvijalca z naslednjimi atributi:

- ime,
- elektronski naslov,
- delovno mesto,
- ime razvojne ekipe,
- status (aktiven/neaktiven).

V primeru, da aplikacija uporablja tako GitHub kot Bitbucket za dostop do repozitorijev, bi lahko razvijalec imel več različnih uporabniških imen, zato so le-ta shranjena v posebni entiteti **DeveloperAccounts (računi razvijalca)**, ki vsebuje naslednje atribute:

- referenca na razvijalca,
- uporabniško ime,
- tip računa (npr. GitHub, Bitbucket).

Project (projekt)

Gre za entiteto, katere namen je združevanje več repozitorijev v skupen projekt. Relacija med projektom in repozitorijem je torej mnogo-mnogo (N:M). V večini primerov je dejansko en repozitorij vezan na en projekt. V nekaterih primerih pa so projekti (še posebej, kadar gre za večje projekte) sestavljeni iz več repozitorijev. Atributi so naslednji:

- ime,
- programsko ime (slug),
- opis,

- slika,
- url,
- status.

RepositoryProvider (ponudnik repozitorija)

Je entiteta s podatki za dostop do repozitorijev, konkretno to pomeni uporabniško ime in geslo ali avtentikacijski žeton za dostop do računa na GitHubu, kjer se repozitoriji nahajajo, in do katerih lahko potem dostopa preko API-ja. Atributi so naslednji:

- ime,
- gostitelj (opcije so npr.: GitHub, BitBucket),
- opis,
- uporabniško ime,
- geslo,
- avtentikacijski žeton za API,
- url,
- vrsta računa (organizacija ali posameznik),
- samostojen projekt (pomeni, da cel račun pripada enemu projektu)

Oznake

Podatke o tehnologijah, ki so uporabljene v določenem repozitoriju, se shranjuje v obliki oznak (angl. tags). Nekatere knjižnice imajo podatke o ključnih besedah in se tudi te nahajajo tu. Oznake tehnologij se delijo na več kategorij, npr. programski jeziki, ogrodja, nameščanje aplikacije itd.

Oznake se hranijo v dveh entitetah: Tags in Taggings. Entiteta Tags je sestavljena iz imena oznake ter števca pojavitev oznake. Taggings pa vsebuje referenco na oznako (angl. tag), referenco na entiteto, na katero je vezana oznaka, kategorijo (navedene zgoraj) ter informacijo o avtorju oznake.

Vse entitete so opremljene tudi z datumom kreiranja ter datumom zadnje posodobitve.

5.1 Testni podatki

Za razvoj in testiranje aplikacije je bilo potrebno pridobiti nabor dovolj velike in raznolike količine testnih podatkov, da lahko pokrijemo čim več scenarijev. Testne podatke smo pridobili tako, da smo poiskali različne odprtokodne spletne aplikacije, zgrajene v Symfony2, Ruby on Rails in NodeJS. Nato smo na GitHubu ustvarili poseben uporabniški račun za našo aplikacijo in naredili kopije (angl. fork) prej omenjenih odprtokodnih projektov.

Poglavje 6

Uporabljene tehnologije

6.1 Ruby on Rails

Ruby je visokonivojski objektno usmerjen jezik, ki ga je sredi devetdesetih razvil Japonec Yukihiro "Matz"Matsumoto v želji, da bi jezik služil človeku in ne zgolj računalniku.

Približno deset let kasneje se je na trgu pojavilo odprtokodno ogrodje Ruby on Rails (v nadaljevanju Rails), namenjeno razvoju modernih spletnih aplikacij. Uporablja ga veliko uspešnih podjetij, kot so Airbnb, Basecamp, GitHub in Kickstarter.

Ogrodje se tako kot tudi sam jezik ponaša z visoko intuitivnostjo, lepo berljivo kodo in položno učno krivuljo. Njegova filozofija temelji na dveh principih, ki omogočita hiter razvoj aplikacij ter enostavno vzdrževanje.

- **Ne ponavljaj se** (angl. Don't repeat yourself oz. krajše DRY) je princip, ki odsvetuje ponavljanje enake ali podobne kode na različnih mestih, kar omogoči lažje vzdrževanje kode ter manjšo verjetnost napak.

- **Dogovor pred nastavitvami** (angl. Convention over configuration) pomeni, da mnoge nastavitve namesto razvijalca postavi ogrodje, zato da se razvijalcu ni potrebno ukvarjati z njimi in jih prilagodi le tam,

kjer je res potrebno.

Poleg tega Rails, tako kot tudi mnoga druga ogrodja, spodbuja uporabo arhitekture REST (angl. Representational State Transfer) in temelji na arhitekturnem vzorcu MVC. V nadaljevanju bomo predstavili implementacijo obeh arhitektur.

6.1.1 REST

REST temelji na arhitekturi tipa odjemalec-strežnik, ki teče preko protokola, ki ne hrani stanja (angl. stateless); tipično je to HTTP. Strežnik in odjemalec si izmenjata predstavitev virov na preprost in performančno nezahteven način (v primerjavi z alternativnimi rešitvami, kot je SOAP [22]).

Z viri označujemo podatke in funkcionalnost aplikacije. Nad njimi se izvajajo preproste operacije CRUD (ustvarjanje, branje, posodabljanje in brisanje), ki se preslikajo v metode protokola HTTP (POST, GET, PUT, DELETE).

Rails vsebuje komponento Active Resource, ki izvaja omenjene preslikave in služi kot ogrodje za upravljanje povezave med entitetami oz. poslovno logiko aplikacije ter spletnimi storitvami REST. Primer: če pošljemo zahtevek spletnemu strežniku 'DELETE /users/5', bo ogrodje Rails to interpretiralo kot zahtevo za brisanje entitete uporabnika, pri katerem ima atribut ID vrednosti 5 [34].

6.1.2 MVC

Organizacija kode poteka po arhitekturnem vzorcu MVC, ki smo ga že razložili v prejšnjih poglavjih, tu pa si bomo podrobneje pogledali delovanje na primeru ogrodja Rails.

Model

Model razširja osnovni razred komponente Active Record. Active Record je komponenta, ki implementira ORM (objektno-relacijsko preslikovanje), kar

pomeni, da skrbi za preslikavo elementov podatkovne baze v objekte, in sicer: tabele v razrede, vrstice v objekte in atribute tabel v atribute razredov. V večini primerov se ena tabela v podatkovni bazi preslika v en model znotraj aplikacije.

Poleg tega komponenta opravlja tudi številne druge funkcije, med katerimi so pomembnejše:

- omogočanje neodvisnosti od podatkovne baze, kar pomeni, da nudi enoten vmesnik ne glede na tip baze,
- upravljanje z relacijami in hierarhijo (dedovanjem) med modeli,
- validacija modelov pred pisanjem v bazo,
- izvajanje osnovnih operacij CRUD na razredih in objektih,
- napredno iskanje objektov z ekspresivnim programskim vmesnikom.

Modeli ne vsebujejo atributov, le-ti so avtomatsko pridobljeni z uporabo tehnik meta programiranja preko definicij tabel, s katerim so povezani.

Nadzornik

Nadzornik je implementiran v komponenti Action Controller. Le-ta predstavlja osnovni razred, ki ga potem razširjajo ostali nadzorniki. Komponenta Action Dispatcher sprocesira zahtevek HTTP in nato posreduje parametre specifični akciji v nadzorniku. Ta vsebuje logiko za manipulacijo z modelom in se povezuje s pogledom. Poleg tega komponenta nadzornika ponuja tudi druge funkcionalnosti, kot npr. upravljanje s sejami, preusmeritvami, predpomnjenje, prikazovanje predlog itd.

Pogled

Pogled običajno predstavljajo datoteke, ki vsebujejo kodo HTML. Razširjene so s podmnožico programskega jezika Ruby, imenovano ERB (angl. Embedded Ruby), ki omogoča enostavne operacije nad podatki v pogledu. Za upravljanje z njimi pa skrbi komponenta Action View.

Komponenti nadzornika in pogleda sta zaradi tesne medsebojne povezanosti združeni v skupno komponento Action Pack, ki zajema celoten tok procesiranja zahtevka spletnega brskalnika ter pošiljanja odgovora aplikacije.

Poleg omenjenih komponent sestavljajo ogrodje Rails še nekatere druge [35]:

- Action Mailer služi pošiljanju elektronskih sporočil.
- Active Support razširja standardno knjižnico jezika Ruby s številnimi uporabnimi razredi in funkcijami ter dobro podporo lokalizaciji, časovnim pasom in testiranju.
- Railties je jedro ogrodja, ki povezuje vse komponente ter skrbi za inicializacijo in zagon aplikacije.

Pomemben del ogrodja so tudi programi, ki se izvajajo v ukazni vrstici in so namenjeni upravljanju z aplikacijo. Med najpogosteje uporabljene spadajo:

- *rails server* - zažene spletni strežnik imenovan WEBrick, ki je že vključen skupaj z jezikom Ruby in se uporablja v razvojnem okolju.
- *rails generate* - uporablja se za generiranje datotek modelov, nadzornikov, pogledov itd., zato da se izognemo pisanju ponavljajoče kode (angl. boilerplate code).
- *rails console* - omogoča interaktiven dostop do aplikacije in manipulacijo objektov preko ukazne vrstice.
- *rake* - je samostojno orodje, sorodno ukazu *make* v Unixu in se pogosto uporablja za administrativna opravila, kot npr. migracije podatkovne baze, izvajanje testov in generiranje dokumentacije.

6.2 Sidekiq

Sidekiq je ogrodje za procesiranje v ozadju, napisano v programskem jeziku Ruby. Odlikuje ga enostavna integracija z ogrodjem Rails ter dobre performančne zmogljivosti [36].

Spletna aplikacija ustvari novo opravilo za procesiranje v ozadju, tako da pokliče odjemalca za Sidekiq, ta pa informacije o opravilu zapiše v podatkovno bazo Redis. Na strežniški strani teče proces, ki obdeluje opravila iz čakalne vrste. Proces naloži ogrodje Rails, tako da imajo opravila dostop do njegovih komponent in API-ja. Vsak tip opravila ima svoj delovni proces, medtem ko se posamezna opravila izvajajo v svoji niti.

Opravila se torej izvajajo sočasno z uporabo več niti na proces, kar pomeni prihranitev časa inicializacije opravila in zmanjšanje porabe pomnilnika. Ruby (natančneje, implementacija Ruby MRI) uporablja GIL (angl. Global Interpreter Lock), kar pomeni, da ne govorimo o pravem paralelizmu, temveč o sočasnosti. Istočasno se ne more izvajati več blokov kode, ampak pohitritev izhaja iz sočasnega izvajanja vhodno-izhodnih operacij [37].

Sidekiq vsebuje tudi spletni vmesnik (Sidekiq WebUI), ki je napisan v ogrodju Sinatra in nam omogoča enostaven pregled nad zgodovino izvajanja opravil ter statističnimi podatki.

6.3 PostgreSQL

PostgreSQL je objektno-relacijski sistem za upravljanje podatkovnih baz. Velja za najnaprednejši odprtokodni sistem za baze in vsebuje mnoge funkcionalnosti, ki jih sicer najdemo le med komercialnimi produkti, kot so DB2 in Oracle [38].

Velik poudarek daje izpolnjevanju standarda ANSI za SQL, enostavni razširljivosti ter hitrim ciklom razvoja - nove različice z novimi funkcionalnostmi so izdane zelo pogosto.

Poleg tega podpira številne podatkovne tipe, omogoča napredno iskanje po tekstovnih poljih - iskanje po polnem besedilu (angl. full-text search) in

striktno zagotavlja integriteto podatkov.

6.4 Bootstrap, jQuery, D3.js

Razvoj na strani odjemalca je v naši aplikaciji precej nezahteven, zato smo uporabili le nekaj osnovnih knjižnic.

Bootstrap je priljubljeno odprtokodno ogrodje, ki predstavlja skupek komponent, ki se pri razvoju spletnih aplikacij na odjemalcu najpogosteje pojavljajo. Zagotavlja kompatibilnost z modernimi spletnimi brskalniki in enostavno prilagoditev mobilnim napravam.

jQuery je knjižnica v jeziku JavaScript, ki olajša manipulacijo dokumentnega objektnega modela (DOM), poenostavi odzivanje na dogodke, klice AJAX, kreiranje animacij in še marsikaj drugega. Je zelo priljubljena knjižnica, saj precej pohitri razvoj ter zmanjša količino kode potrebne za razvoj funkcionalnosti.

D3.js je knjižnica, ki se uporablja za risanje grafov. Uporabili smo še dodatno knjižnico d3pie.js, ki je namenjena risanju naprednih krožnih diagramov z veliko nastavljivimi opcijami.

6.5 Heroku

Aplikacijo smo objavili na Heroku, ki je ponudnik PaaS (angl. Platform as a service). Gre za kategorijo oblačnega računalništva, ki kot storitev ponuja platformo za razvoj, izvajanje in upravljanje spletnih aplikacij brez kompleksnosti postavljanja in vzdrževanja infrastrukture [39].

Heroku samodejno zazna programski jezik in ogrodje aplikacije ter na podlagi tega ve, kako upravljati z aplikacijo. V posebnih ali bolj kompleksnih primerih pa lahko z datoteko Procfile natančneje definiramo procese, ki jih mora Heroku izvajati.

Za objavo aplikacije se najpogosteje uporablja git. Heroku ustvari repozitorij na strežniku, katerega razvijalec potem doda v git kot oddaljen re-

pozitorij (angl. remote) in običajno poimenuje ‘heroku’. Nato s preprostim ukazom *git push heroku master* sproži proces objave nove različice aplikacije [40].

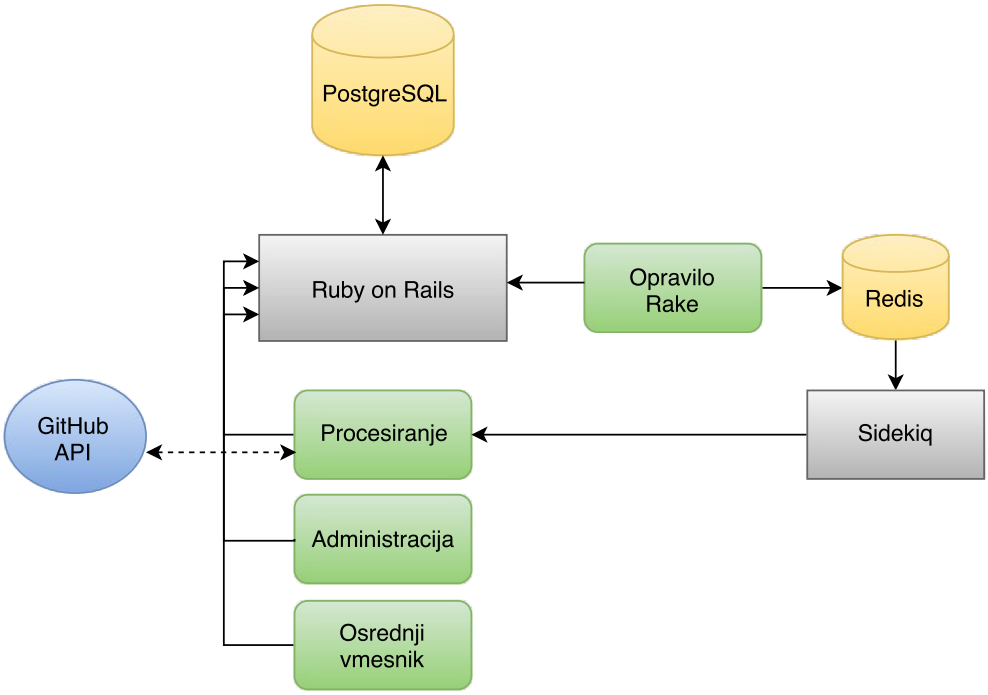
Poglavje 7

Arhitektura in razvoj funkcionalnosti

Aplikacija je v osnovi zgrajena iz treh komponent:

- **procesiranje v ozadju** - delovni proces, ki se izvaja v ozadju in je namenjen črpanju podatkov iz GitHuba ter posodabljanju podatkovne zbirke,
- **osrednji vmesnik** - jedro aplikacije z grafičnim vmesnikom, preko katerega uporabnik lahko dostopa do podatkov,
- **administrativni vmesnik** za upravljanje določenih podatkov, ki niso pridobljeni avtomatsko.

V nadaljevanju si bomo pobližje pogledali vse tri komponente, s poudarkom na procesiranju v ozadju, ki je pomembnejši in kompleksnejši del aplikacije. Slika 7.1 prikazuje posamezne komponente aplikacije in komunikacijo med njimi.



Slika 7.1: Prikaz arhitekture aplikacije.

7.1 Procesiranje v ozadju

Namen procesiranja v ozadju je izgradnja podatkovne zbirke, ki sloni na podatkih pridobljenih iz zunanjih virov, kar so načeloma ponudniki gostovanja repozitorijev. V našem primeru smo implementirali pridobivanje podatkov iz GitHuba.

Ažurnost podatkov dosežemo z uporabo razporejevalnika opravil (angl. Cron job). Alternativna rešitev bi bila uporaba tako imenovanih "webhookov", ki omogočajo dostavo obvestil o spremembah, tako da ob določenih dogodkih sprožijo zahtevek na našo spletno storitev. Takšen način je nekoliko optimalnejši, vendar tudi kompleksnejši za izvedbo. V naši aplikaciji ne potrebujemo takojšnjih posodobitev, zato smo se odločili, da je posodabljanje enkrat dnevno ustrezno.

Opravilo mora prepoznati in vnesti nove repozitorije ter poskrbeti za posodobitev obstoječih repozitorijev s svežimi podatki o razvijalcih, knjižnicah

in uporabljenih tehnologijah. Posodabljanje repozitorija je časovno zahtevna operacija in večinoma neodvisna od izvajanje posodobitev drugih repozitorijev, zato smo se odločili za večnitno procesiranje.

7.1.1 Opravilo v orodju Rake

Opravilo posodabljanja podatkov se izvaja v orodju Rake, ki je del ogrodja Rails in uporablja domensko specifično nadgradnjo jezika Ruby, ki omogoči enostavno definiranje in upravljanje z opravili. Spodaj je primer nekoliko poenostavljene kode, ki se izvede ob klicu ukaza *rake data:import*.

```
namespace :data do
  desc "Import repository data"
  task import: :environment do
    RepositoryProvider.all.each do |provider|
      client = Importer::Client::BaseClient.create(provider) || return
      client.load_repositories.each do |r|
        Importer::RepositoryImporter.perform_later(provider.id, r[:
            full_name])
      end
    end
  end
end
```

Postopek izvajanja naloge je sledeč:

1. V podatkovni bazi morajo biti definirani ponudniki repozitorijev, ki vsebujejo podatke za dostop do repozitorija.

2. Aplikacija iterira čez vse ponudnike repozitorijev in glede na podatek o gostitelju ustvari novo instanco odjemalca za komunikacijo s spletno storitvijo gostitelja, kot je GitHub.

3. Odjemalec pošlje spletni storitvi gostitelja zahtevek za seznam vseh repozitorijev.

4. Aplikacija za vsak repozitorij iz seznama sproži proces posodabljanja. Podatek o imenu repozitorija in referenci ponudnika repozitorija se

shrani v podatkovno bazo Redis, ki nato sporoči Sidekiqu, naj izvede proces. Gre za mehanizem objavi/naroči (angl. publish/subscribe oz. krajše pub/sub), katerega namen je komunikacija sporočil med različnimi nepovezanimi komponentami sistema, ki ne vedo ena za drugo [41].

7.1.2 Razvoj odjemalca za dostop do repozitorijev

Naloga odjemalca je pridobivanje podatkov o repozitorijih iz zunanjih virov. Arhitektura je zastavljena tako, da je mogoče v sistem enostavno dodajati tudi nove odjemalce, ki dostopajo do drugih ponudnikov gostovanja repozitorijev, kot so npr. BitBucket ali Gitlab.

Osnovni razred, ki predstavlja odjemalca, se imenuje BaseClient, ostali odjemalci pa razširjajo ta razred.

```
class BaseClient
  def self.create(provider)
    case provider.host
      when RepositoryProvider::GITHUB
        client = Client::GitHubClient.new(provider)
      when RepositoryProvider::BITBUCKET
        client = Client::BitbucketClient.new(provider)
      else
        raise "Provider #{provider.host} is not implemented!"
    end
    client
  end

  def initialize(provider)
    @provider = provider
  end
end
```

Po principu načrtovalskega vzorca *Factory*, metoda *create* vrne novo instanco razreda glede na podani argument. V tem primeru argument predstavlja ime gostitelja (npr. GitHub), instanca pa je objekt razreda odjemalca,

ki implementira dostop do tega gostitelja.

Odjemalec nato pošilja zahtevke vmesniku API, odgovore pa pretvori v obliko, ustrezno našemu podatkovnemu modelu. Ideja je torej, da vsi odjemalci vračajo podatke v enotni obliki (po načrtovalskemu vzorcu *Adapter*).

Implementacija odjemalca za GitHub

GitHub API je implemetiran v arhitekturnem stilu REST (pravimo, da je RESTful). Komunikacija poteka preko protokola HTTP, viri so med seboj hipertekstovno povezani, podprti so različni formati za prenos, med katerimi je najpogostejši JSON (angl. JavaScript Object Notation). Gre za format, ki temelji na dveh strukturah: zbirka parov ključ/vrednost ter urejen seznam. Ker so te strukture znane v večini modernih jezikov, je zato zelo enostaven za uporabo in lahko berljiv [42].

Uporabili smo Octokit, uradno knjižnico GitHuba za jezik Ruby, ki zelo poenostavi klice na njihov API.

Podprtih je več možnih načinov avtentikacije. V naši aplikaciji smo podprli avtentikacijo z uporabo žetona OAuth. GitHub omogoča enostavno upravljanje z avtentikacijskimi žetoni preko vmesnika na spletu. Žetonu je mogoče določiti tudi nivoje dostopa in tako omejiti dostop do določenih podatkov. Avtentikacija je z uporabo knjižnice Octokit enostavna:

```
@octokit = Octokit::Client.new(:access_token => "<avtentikacijski zeton>
    ")
```

Spodnji primer prikazuje dostop do seznama vseh privatnih repozitorijev, ki pripadajo računu določene organizacije:

```
repositories = @octokit.org_repos(@provider.name, {
  :per_page => 10, :type => 'private'
})
```

API vrne rezultat v formatu JSON, knjižnica pa ga pretvori v podatkovni tip seznama, po katerem lahko iteriramo. Ker je API hipertekstovno povezan, lahko enostavno dostopamo tudi do drugih virov. Spodnji primer prikazuje,

kako dobimo podatke o razvijalcih in število njihovih prispevkov:

```
contributors = repository.rels[:contributors].get.data
```

7.1.3 Implementacija branja podatkov o knjižnicah

Spletne aplikacije običajno uporabljajo programe za upravljanje paketov, ki skrbijo za nadzor nad knjižnicami, tako da v posebni datoteki hranijo seznam vseh nameščenih knjižnic ter njihovih različic.

Vsak programski jezik ima svoj program za upravljanje s paketi. V osnovi vsi opravljajo enako nalogo, vendar pa so razlike v implementaciji. V kolikor torej želimo dobiti seznam uporabljenih knjižnic na posameznem projektu oz. repozitoriju, moramo upoštevati programski jezik in specifike upravljalca paketov za ta jezik.

Funkcionalnost smo zasnovali tako, da je dodajanje podpore novim programskim jezikom oz. paketnim upravljalcem enostavno. Za začetek pa smo implementirali podporo za tri različne jezike oz. upravljalce:

- PHP (Composer),
- Ruby (Bundler),
- Node.js (NPM - Node Package Manager).

Arhitektura razredov je podobna kot pri odjemalcih. Imamo torej osnovni razred za branje knjižnic, ki ga razširjajo razredi za specifični jezik.

PHP in Composer

Composer uporablja za upravljanje s paketi dve datoteki: composer.json, v kateri hrani ime knjižnice in njeno variabilno različico ter composer.lock, v kateri hrani podatke o dejanskih nameščenih različicah in podrobnejše podatke o paketih oz. knjižnicah.

Ukaz *composer install* prebere imena in različice knjižnic iz datoteke composer.lock ter namesti tiste, ki še niso nameščene. Ukaz *composer update* pa

prebere datoteko composer.json in namesti knjižnice, ki se še ne nahajajo v datoteki composer.lock, ali pa posodobi knjižnice, kjer je mogoče namestiti novejšo različico.

Naša aplikacija prebere podatke o knjižnicah iz datoteke composer.lock, saj je v njej shranjenih več podatkov o paketu: ime knjižnice, različica, avtorji, odvisne knjižnice, tip licence, opis, ključne besede in še mnogi drugi.

```
class PhpReader < BaseReader
  FILE = 'composer.lock'
  LANGUAGE = 'php'

  def get_libraries
    content = JSON.parse(get_file_content, :symbolize_names => true)
    content[:packages].map do |package|
      package[:url] =  package[:source] && package[:source][:url]
      ? package[:source][:url]
      : nil
      package = package.slice(:name, :version, :keywords, :homepage, :
          url, :description)
      package[:language] = LANGUAGE
      package[:slug] = LANGUAGE + '/' + package[:name]
      package
    end
  end
end
```

Node.js in NPM (Node Package Manager)

Node.js shranjuje podatke o paketih oz. knjižnicah v datoteki package.json, vendar le ime in različico paketa. Več podatkov o paketu je mogoče dobiti z zahtevkom na API registra paketov, ki nam v formatu JSON vrne zelo podrobne informacije o paketu.

```
class NodeJsReader < BaseReader
  FILE = 'package.json'
  LANGUAGE = 'nodeJS'

  def get_libraries
    packages = JSON.parse(get_file_content, :symbolize_names => true)[:
        dependencies]
    packages.map do |package_name, package_version|
      response = Net::HTTP.get_response(URI.parse("http://registry.npmjs
          .org/#{package_name}"))
      result = JSON.parse(response.body, :symbolize_names => true)
      {
        :name => package_name,
        :version => package_version,
        :language => LANGUAGE,
        :slug => LANGUAGE + '/' +  package_name,
        :description => result[:description],
        :homepage => result[:homepage],
        :url => result[:repository][:url],
        :keywords => result[:keywords]
      }
    end
  end
end
```

Ruby in Bundler

Datoteka Gemfile vsebuje seznam knjižnic. Bundler z ukazom *bundle install* datoteko prebere, namesti pakete in nato v Gemfile.lock zapiše podrobnejše podatke o nameščenih paketih. Naša aplikacija datoteko prebere in iz nje izlušči podatke o nameščenih knjižnicah. Ker datoteka ni v formatu JSON, smo si pomagali z regularnimi izrazi.

Podobno, kot pri branju paketov Node, mora aplikacija narediti zahtevek na repozitorij Rubygems.org, ki potem v formatu JSON vrne podrobnejše informacije o paketu.

Slabost paketnega upravljalca v Rubyju je, da za razliko od upravljalcev

Composer in NPM, ne vsebuje ključnih besed za knjižnico, kar pomeni, da v tem primeru iskanje po njih v aplikaciji ne bo mogoče.

7.1.4 Implementacija prepoznavanja tehnologij

Poleg knjižnic, ki so uporabljene na projektu, smo želeli dobiti tudi informacije o drugih tehnologijah: programskih jezikih, ogrodjih, podatkovnih bazah, infrastrukturi itd. Veliko teh informacij se nahaja v izvorni kodi projekta, zato lahko do njih pridemo tako, da poiščemo ustrezne datoteke in nize znotraj teh datotek.

Naloge smo se lotili tako, da smo na vzorcu testnih podatkov preučili strukturo datotek v različnih ogrodjih in tako prišli do seznama tehnologij, ki jih je na podlagi izvorne kode mogoče razbrati. Tehnologije smo razvrstili v več kategorij in v vsaki kategoriji podprli nekaj pogosto uporabljenih tehnologij.

- **Programski jeziki oz. okolja**: PHP, Ruby, NodeJS.
- **Ogrodja**: Symfony2, Ruby on Rails, Express.js.
- **Podatkovne baze**: PostgreSQL, MySQL, Redis, Elasticsearch.
- **JavaScript**: Grunt, gulp, Bower, webpack, Ember, Backbone.js, React, Babel, AngularJS.
- **Zvezna integracija**: Travis CI, CircleCI.
- **Testiranje**: PHPUnit, Behat, PHPspec, Selenium, RSpec, Capybara, PhantomJS.
- **Infrastruktura**: Docker, Vagrant, Chef, Heroku.
- **Nameščanje**: Mina, Capistrano.
- **Drugo**: Scrutinizer, Sidekiq, Mandrill, Mailchimp.

Želeli smo, da je v kodi dobro razviden pregled nad obstoječimi podprtimi tehnologijami ter enostavno dodajanje novih tehnologij. Odločili smo se, da pravila za iskanje tehnologij definiramo v standardu YAML, saj gre za človeku zelo prijazno obliko strukturiranja podatkov, ki je nadmnožica formata JSON (vsebuje vse njegove funkcionalnosti).

V vsaki kategoriji je definiranih več tehnologij, za vsako od njih pa imamo definiranih več pogojev, od katerih mora biti vsaj en izpolnjen, da je tehnologija prepoznana. V nekaterih primerih zadostuje že, da obstaja specifična datoteka, v drugih primerih pa je potrebno iskanje z regularnimi izrazi po vsebini datoteke. Spodaj je izsek iz datoteke s pravili za iskanje tehnologij.

```
# DEPLOY
deploy:
  mina:
    file: "Gemfile"
    pattern:  "gem 'mina'|gem \"mina\""

  capistrano:
    file: "Capfile"

# DATABASE
  database:
    postgresql:
    - file: "config/database.yml"
      pattern:  "adapter: postgresql"

    - file: ".travis.yml"
      pattern: "postgres"

    - file: "Gemfile"
      pattern: "gem 'pg'|gem \"pg\""
```

7.1.5 Obvladovanje sočasnosti

Sidekiq omogoča paralelno procesiranje, kar v našem primeru pomeni, da za vsak repozitorij ustvari novo nit, v kateri izvaja procesiranje.

Podatki med posameznimi repozitoriji so večinoma med seboj neodvisni, obstaja pa nekaj izjem, kjer lahko sočasnost povzroči neželene rezultate.

Ko odjemalec vrne seznam vseh knjižnic, ki so uporabljene v repozitoriju, aplikacija preveri, če določena knjižnica že obstaja v našem sistemu. V primeru, da je še ni, ustvari nov objekt in ga shrani v bazo. Enako velja tudi pri ustvarjanju novih objektov razvijalcev in oznak tehnologij. Lahko se zgodi, da dve niti istočasno poskusita ustvariti isti objekt (angl. race condition), kar pa sproži izjemo, ker je na polju v podatkovni bazi nastavljen unikatni indeks. Izjema povzroči zaustavitev procesiranja, zato jo je potrebno ujeti.

Za dodajanje oznak uporabljamo knjižnico, ki ni napisana s podporo sočasnosti, zato smo morali del knjižnice prepisati z lastno metodo, ki prepreči, da bi prišlo do zaustavitve procesiranja.

```
def self.find_or_create_tag(tag_name, list)
  retries ||= 2
  comparable_tag_name = comparable_name(tag_name)

  existing_tag =
    ActsAsTaggableOn::Tag
      .named_any(list)
      .detect { |tag| comparable_name(tag.name) == comparable_tag_name }

  existing_tag || ActsAsTaggableOn::Tag.create(:name => tag_name)

rescue ActiveRecord::RecordNotUnique
  retry unless (retries -= 1).zero?
end
```

7.1.6 Prikaz delovanja

Opravilo se načeloma izvede enkrat dnevno v razporejevalniku opravil, lahko pa ga sprožimo tudi ročno. V ukazni vrstici poženemo *rake data:import* in aplikacija izpiše, kateri repozitoriji bodo posodobljeni.

```
Loading repositories for provider: colibri-data - github
Repository colibri-data/clubmaster: scheduled for import.
```

```
Repository colibri-data/cqrs-php-sandbox - skipping, nothing new.
Repository colibri-data/david-www - skipping - nothing nothing new.
Repository colibri-data/diamantedesk-application - scheduled for import.
```

Nato zaženemo ukaz *bundle exec sidekiq -c 10*, ki ustvari deset niti in prične s procesiranjem. V konzoli se izpiše:

```
Importer::RepositoryImporter JID-dfe1398083bb265e4ecde075 INFO: start
Importer::RepositoryImporter JID-fb270af97ee377126efd4fb3 INFO: start
Importer::RepositoryImporter JID-fb270af97ee377126efd4fb3 INFO: done:
    15.988 sec
Importer::RepositoryImporter JID-730f5940668d636a3d07d54b INFO: start
Importer::RepositoryImporter JID-dfe1398083bb265e4ecde075 INFO: done:
    16.893 sec
```

Primerjali smo čas zaporednega in paralelnega izvajanja. Pri procesiranju 15 repozitorijev je bil čas zaporednega izvajanja približno 5 minut, paralelnega pa le 1 minuto. Iz tega je razvidno, da je uporaba tehnologije Sidekiq pohitrila izvajanje za faktor 5.

7.2 Administracijski vmesnik

7.2.1 Razvoj

Dostop do administracijskega vmesnika je omogočen le uporabnikom s posebnim dovoljenjem. Njegov namen je upravljanje s podatki ponudnikov repozitorijev in odpravljanje napak, ki so se pojavile pri uvozu podatkov.

Uvoz podatkov načeloma poteka avtomatsko z eno izjemo, in to je povezovanje repozitorijev s projekti. Privzeto se za vsak repozitorij ustvari povezan projekt, ki prevzame ime in opis repozitorija, razen če ima ponudnik repozitorija omogočeno nastavitev, da se vsi njegovi repozitoriji povežejo na en projekt. Vendar pa obstajajo tudi izjeme, npr. en račun ima lahko več repozitorijev, ki so del enega projekta. Administrator lahko popravi napake, ki so nastale pri uvozu podatkov. Prav tako lahko določene repozitorije onemogoči, če so za uporabnike aplikacije nerelevantni.

Urejati je mogoče tudi druge podatke, ki niso bili pravilno prepoznani, ali pa so bili izpuščeni. Potrebno bi bilo razviti še funkcionalnost, ki preprečuje, da bi bili z vnovičnim procesiranjem ročno popravljeni podatki, prepisani.

Uporabljena je knjižnica ActiveAdmin, ki zna na podlagi modelov avtomatsko zgenerirati vmesnik za operacije CRUD. Za vsak model obstaja razred, ki služi kot navodila knjižnici za prilagoditev prikaza in upravljanja s tem modelom. Uporablja se DSL, ki prilagajanje zelo poenostavi in ga približa naravnemu jeziku.

Naslednji primer prikazuje prilagoditev administracije projektov. Vsebuje definicijo parametrov, ki jih uporabnik lahko spreminja, prilagojen je izpis posameznega projekta, da izpiše tudi povezane repozitorije ter izpis seznama projektov, ki izpiše samo definirane atribute.

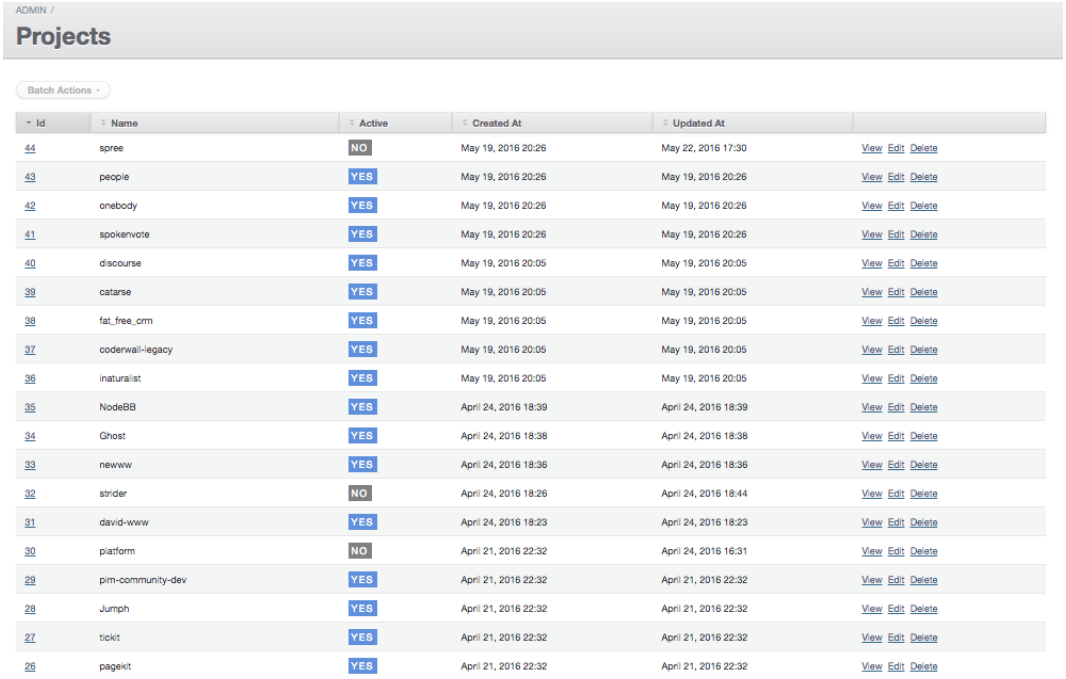
```
ActiveAdmin.register Project do
  permit_params :name, :description, :url,  :active, repository_ids: []

  show do
    attributes_table :id, :name, :description, :url, :active
    panel "Repositories" do
      table_for project.repositories do
        column :name
        column :primary_language
        column :primary_framework
      end
    end
  end

  index do
    id_column
    column :name
    column :active
    column :created_at
    column :updated_at
    actions
  end
end
```

7.2.2 Prikaz delovanja

Administracijski vmesnik je precej obsežen in vsebuje veliko podstrani, ki pa so si med seboj zelo podobne, saj gre za operacije CRUD na različnih entitetah. V nadaljevanju sta prikazani dve podstrani vmesnika.

Seznam projektov

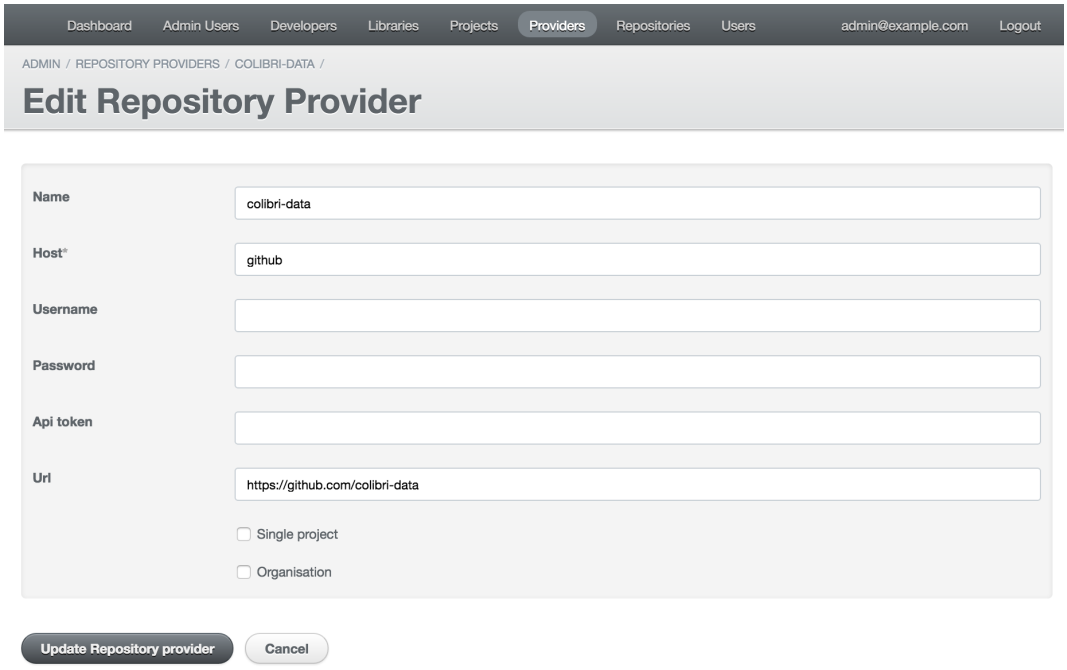
Seznam vseh projektov je prikazan na sliki 7.2 in vsebuje osnovne informacije o projektu (ime, status, datum kreiranja in datum zadnje posodobitve) ter gumbe, ki omogočajo prikaz podrobnejših podatkov, urejanje ali brisanje projekta.



Slika 7.2: Seznam projektov.

Urejanje ponudnika repozitorija

Urejanje ponudnika repozitorija je omogočeno preko obrazca, ki je prikazan na sliki 7.3. Pri urejanju ali dodajanju ponudnika repozitorija je obvezno izpolniti sledeča polja: ime, tip gostitelja, avtentikacijski podatki (žeton ali pa uporabniško ime in geslo) ter spletni naslov. Poleg tega lahko administrator odkljuka polje, ki označuje, da gre za račun organizacije in ne posameznika, ter polje, ki označuje, da vsi repozitoriji na tem naslovu pripadajo enemu projektu.



Slika 7.3: Urejanje ponudnika repozitorija.

7.3 Osrednji vmesnik

7.3.1 Razvoj

Uporabniški vmesnik je jedro aplikacije, kjer lahko uporabnik dostopa do vseh informacij, ki mu lahko potencialno olajšajo delo, ali pa razširijo njegovo znanje.

V ogrodju Rails smo implementirali naslednje nadzornike:

- **ProjectsController** - prikaz seznama projektov in profilne strani projekta,
- **IndexController** - prva stran,
- **SearchController** - rezultati iskanja,
- **LibraryController** - profilna stran knjižnice in dodajanje komentarja h knjižnici.

Za implementacijo uporabniškega sistema smo uporabili knjižnico Devise. Je modularna in fleksibilna rešitev, ki ponuja širok nabor funkcionalnosti, kot so kriptiranje gesla, avtentikacijo, ponastavitev gesla, registracijo, pošiljanje elektronskih sporočil, pomnjenje prijave s piškotkom itd. Knjižnico smo razširili tako, da lahko uporabnik pri registraciji poleg osnovnih podatkov vnese tudi druge, kot so ime, priimek, ime razvoje skupine in delovno mesto. Aplikacijo lahko uporablja tudi anonimni uporabnik, vendar pa nima možnosti komentiranja knjižnice.

Iskalnik smo implementirali z uporabo funkcionalnosti iskanja po polnem besedilu, ki ga omogoča podatkovna baza PostgreSQL in je precej bolj učinkovito kot iskanje z uporabo ukaza *LIKE*. Deluje tako, da znotraj tekstovnega polja poišče korene besed (z uporabo angleškega slovarja) ter izloči nepomembne besede.

Koda spodaj prikazuje poizvedbo za iskanje po knjižnicah.

```
@libraries = Library.find_by_sql(["
  SELECT
    DISTINCT(l.*)
  FROM
    (SELECT
      id, slug,  name,  language, description,
      to_tsvector(name) || to_tsvector(description) AS library_text
    FROM
      libraries
    ) AS l
  LEFT JOIN
    taggings tt ON taggable_id = l.id AND taggable_type = 'Library'
  LEFT JOIN
    tags t ON tt.tag_id = t.id
  WHERE
    l.library_text @@ to_tsquery(:query)
  OR
    t.name = :query
  ",
  {query: params[:query]}
])
```

Z izjemo iskanja in prikazovanja nekaterih statistik, smo se pri pisanju poizvedb jeziku SQL izognili ter uporabili vmesnik komponente Active Record. Spodnja koda prikazuje poizvedbo za seznam projektov:

```
def index
  @projects = Project.includes(
    repositories: [
      contributions: [:developer],
      library_versions: [:library],
      taggings: [:tags]
    ]
  )
  .where(active: true, repositories: {active: true})
end
```

Spodaj je primer kode za nadzornik LibrariesController, ki prikazuje poizvedbo za prikaz posamezne knjižnice ter dodajanja komentarja h knjižnici.

```
class LibrariesController < ApplicationController
  def show
    @library =
      Library
      .includes(:repositories)
      .find_by(slug: params[:slug])
  end

  def comment
    library = Library.find_by(slug: params[:slug])

    LibraryComment.create(
      comment: params[:comment],
      recommendation: params[:recommendation],
      library_id: library.id,
      user_id: current_user.id,
    )

    redirect_to :back, flash: {comment_msg: 'Thanks for your input!'}
  end
end
```

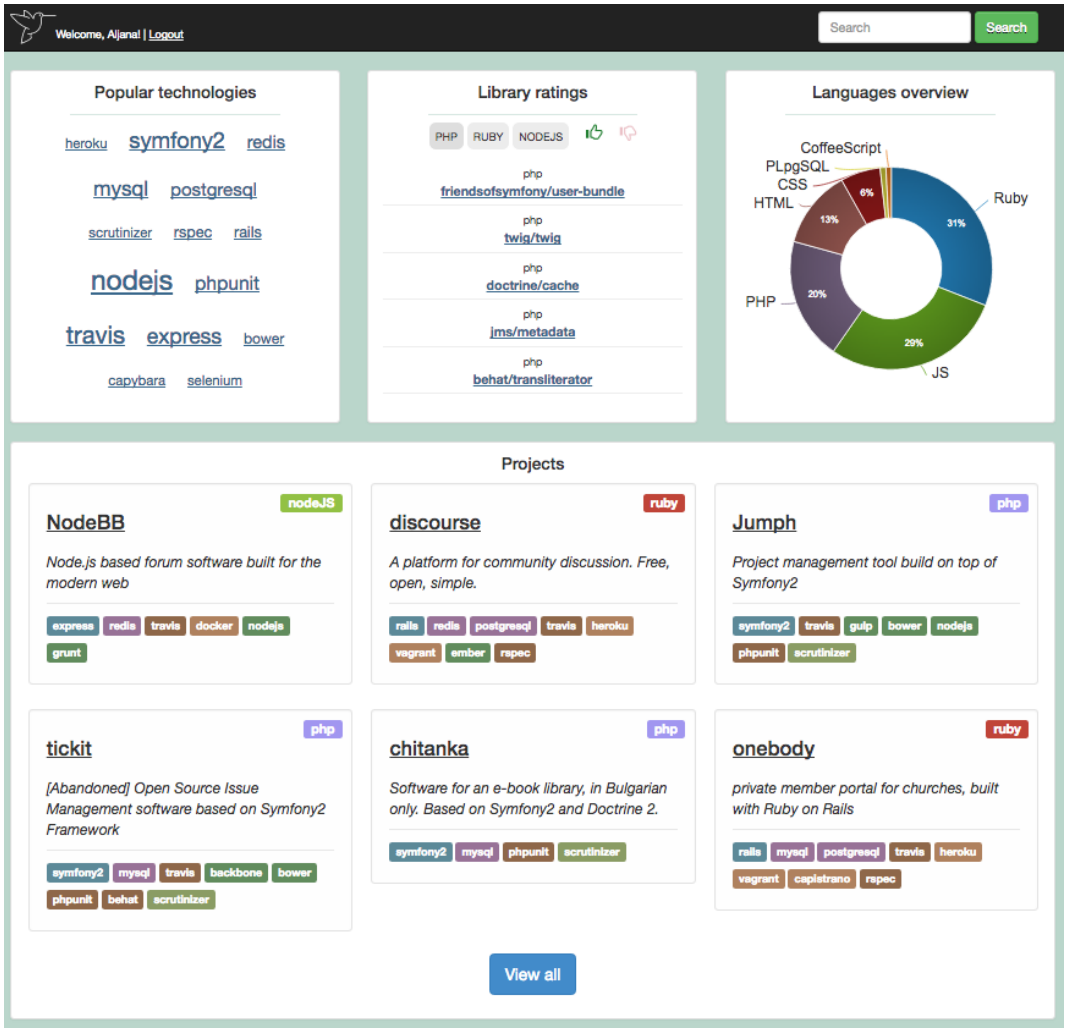
7.3.2 Prikaz delovanja

Vstopna stran

Vstopna stran uporabniku postreže z nekaj zanimivimi informacijami o projektih, ki se v podjetju razvijajo. Kot je razvidno iz slike 7.4, vsebuje naslednje komponente:

- "Oblak" priljubljenih tehnologij. Število projektov oz. repozitorijev, kjer je tehnologija uporabljena, določa velikost pisave.
- Seznam najbolj in najmanj priljubljenih knjižnic glede na ocene uporabnikov aplikacije in filtriranje glede na programski jezik.

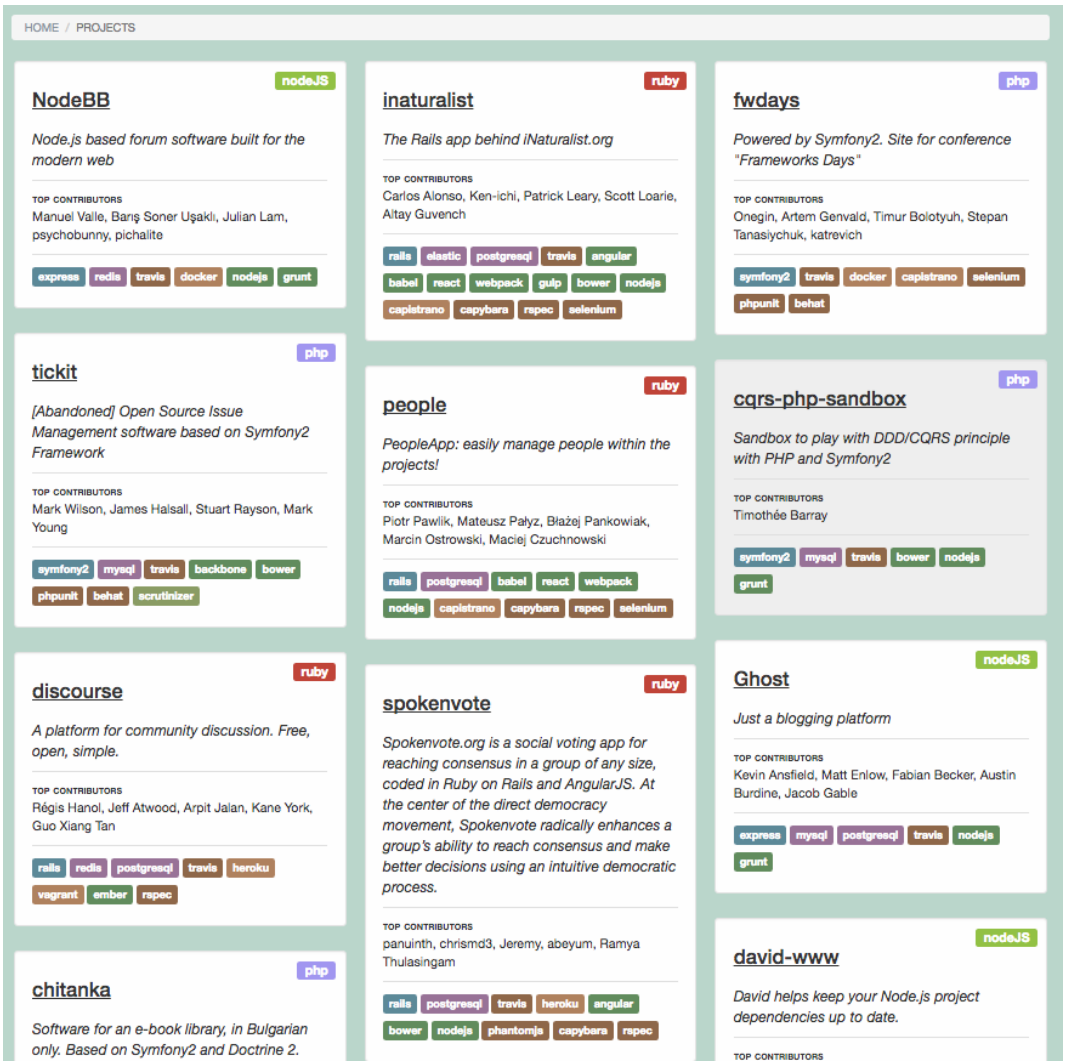
- Krožni diagram s statistiko uporabe programskih jezikov.
- Seznam projektov, ki so bili nazadnje posodobljeni s povezavo na profilno stran projekta ter seznam vseh projektov.



Slika 7.4: Vstopna stran.

Seznam projektov

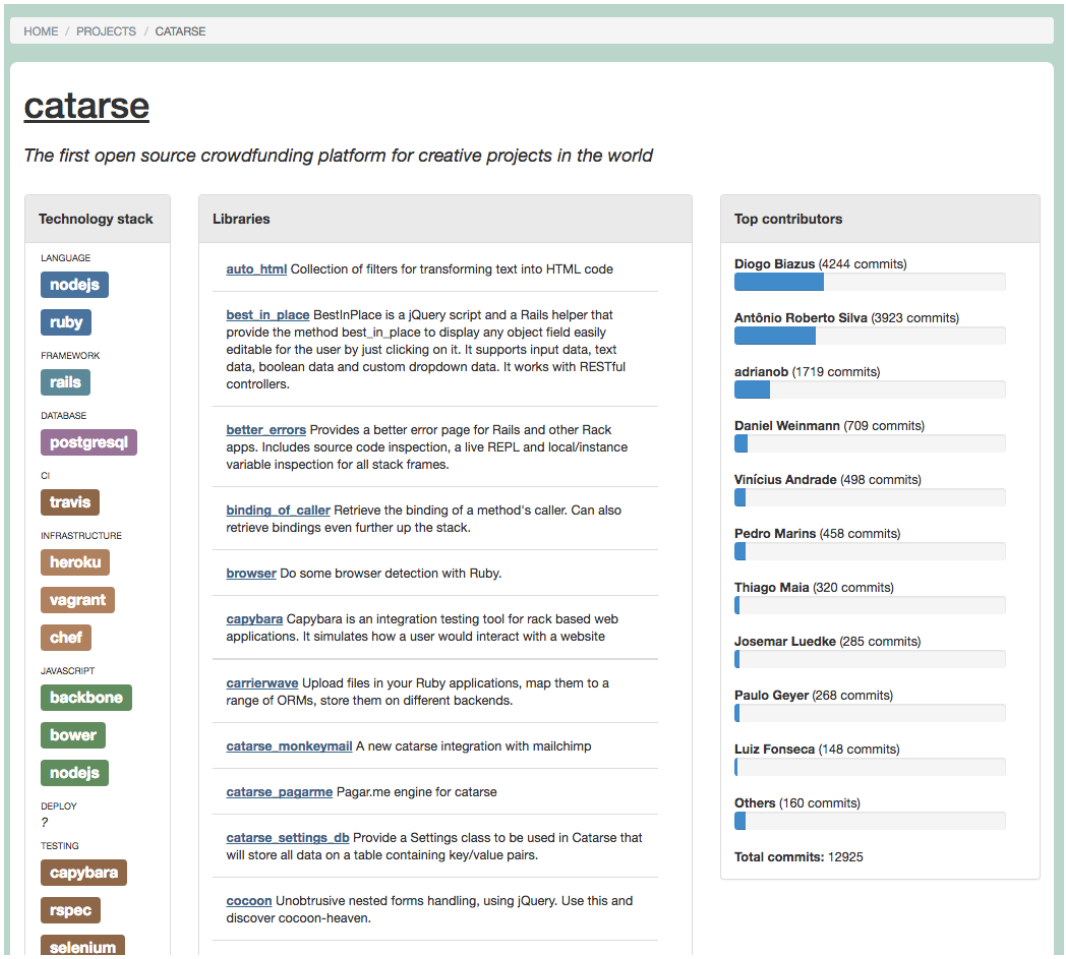
Na sliki 7.5 vidimo prikazan seznam vseh projektov z naslednjimi podatki: ime projekta, kateremu sledi njegov opis, ter programski jezik v zgornjem desnem kotu, v spodnjem delu pa se nahajajo imena najaktivnejših sodelujočih razvijalcev in uporabljene tehnologije. S klikom na ime projekta uporabnik pride na profilno stran projekta.



Slika 7.5: Seznam projektov.

Profilna stran projekta

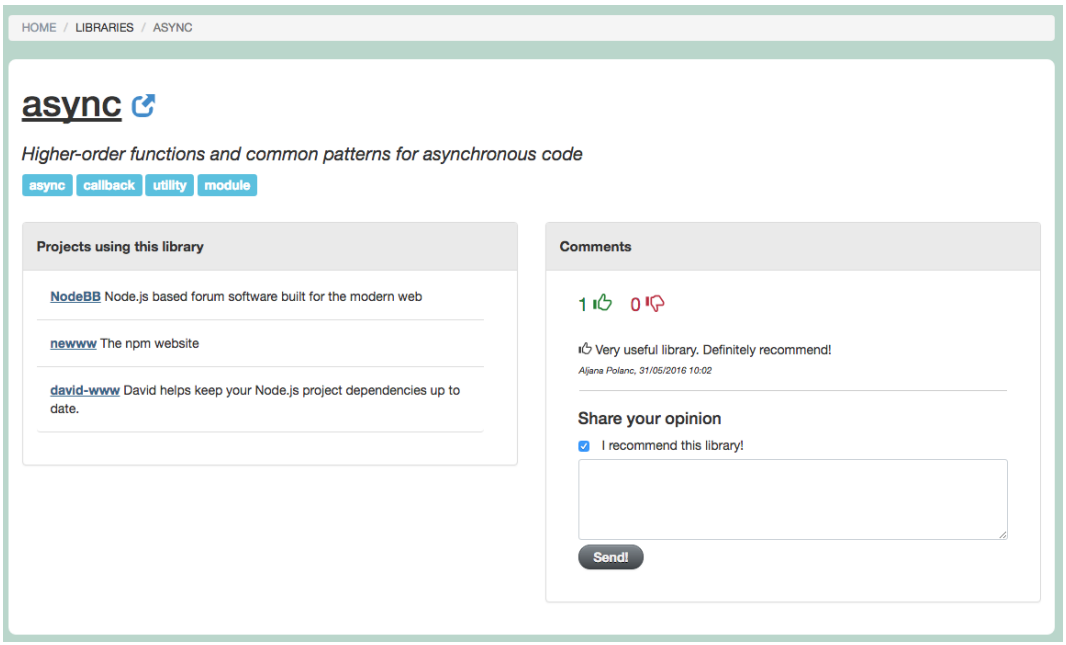
Profilna stran projekta prikazuje podrobnejše informacije o projektu, kar lahko vidimo na sliki 7.6. Na levi strani so tehnologije razvrščene po kategorijah. Sredinski del zavzemajo podatki o knjižnicah z imenom, opisom in povezavo na profilno stran knjižnice. Desni gradnik pa prikazuje razvrstitev razvijalcev na projektu glede na število njihovih prispevkov.



Slika 7.6: Profilna stran projekta.

Profilna stran knjižnice

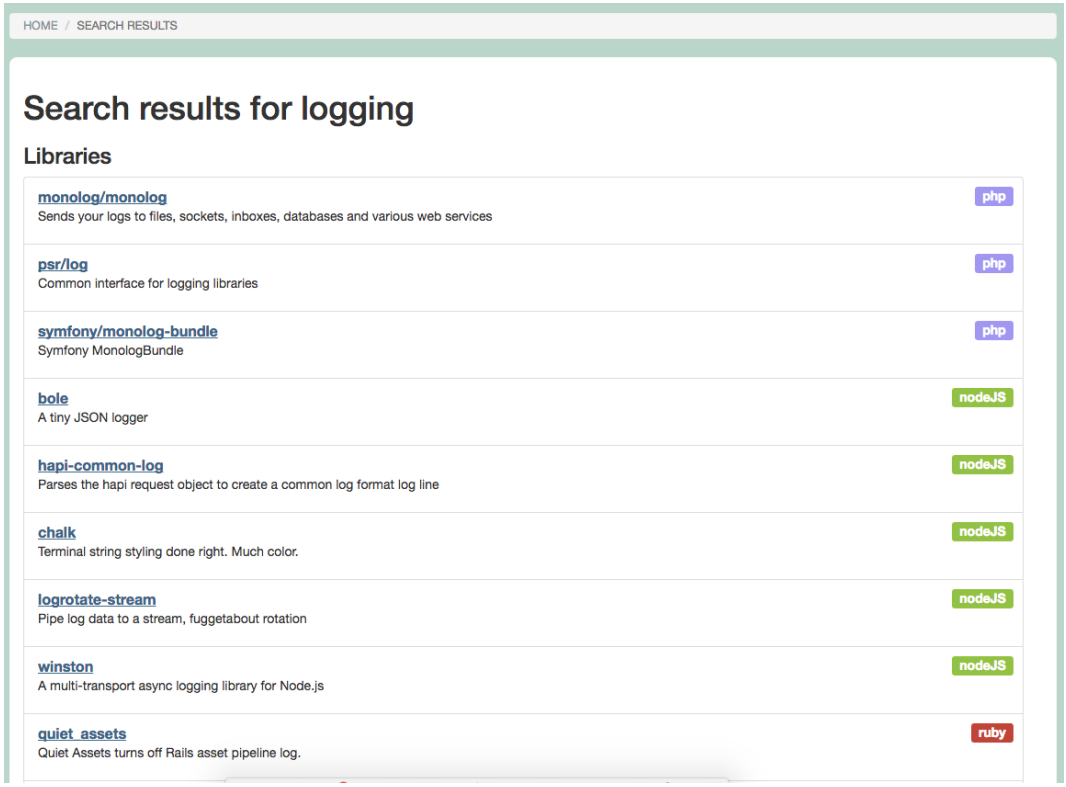
Profilna stran knjižnice vsebuje ime knjižnice, opis, oznake in povezavo na repozitorij na GitHubu, kar je prikazano na sliki 7.7. Na levi strani je seznam vseh projektov, kjer je knjižnica uporabljena. Na desni pa so komentarji, ki so jih uporabniki podali ter število priporočil oz. odsvetovanj. Spodaj je obrazec, preko katerega lahko uporabnik priporoča oz. odsvetuje knjižnico ter zraven dopiše komentar.



Slika 7.7: Profilna stran knjižnice.

Rezultati iskanja

Uporabnik lahko išče po knjižnicah (ime, opis, oznake) in projektih (ime, opis, programski jezik, ogrodje, druge tehnologije). Primer prikaza rezultatov iskanja je na sliki 7.8.



Slika 7.8: Rezultati iskanja.

Poglavje 8

Sklepne ugotovitve

V diplomskem delu smo razvili aplikacijo, namenjeno podjetjem, ki se ukvarjajo z razvojem spletnih aplikacij v programskih jezikih PHP, Ruby in okolju NodeJS. Namen aplikacije je omogočiti pregled nad tehničnimi in drugimi značilnostmi projektov, ki se v podjetju razvijajo ali vzdržujejo.

Glavna značilnost naše aplikacije je, da so podatki o projektih pridobljeni avtomatsko, tako da se aplikacija poveže s spletno storitvijo GitHuba, kjer se nahajajo repozitoriji projektov. Nato z branjem določenih datotek v repozitoriju ali iskanjem določenih nizov oz. vzorcev nizov znotraj datotek, razbere tehnične značilnosti projekta, kot so programski jeziki, ogrodja, knjižnice, podatkovne baze itd. S tem se izognemo potrebi po ročnem vzdrževanju podatkov, ki je zamudno, obenem pa nam avtomatsko dnevno posodabljanje podatkov zagotavlja ažurnost ter zmanjšuje verjetnost, da bi prišlo do napake.

Aplikacija uporabniku omogoča naslednje funkcionalnosti:

- pregled najbolj uporabljenih programskih jezikov,
- pregled najbolj uporabljenih drugih tehnologij,
- pregled najbolj in najmanj priljubljenih knjižnic, razvrščenih glede na programski jezik,
- oddajo pozitivnega ali negativnega odziva na določeno knjižnico,

- pregled projektov,
- pregled posamezne knjižnice,
- iskanje po knjižnicah in projektih,
- administrativni vmesnik za popravljanje nepravilnosti podatkov oz. vnašanje nekaterih drugih podatkov.

Pri razvoju smo si zastavili dva glavna cilja. Prvi je bil razviti aplikacijo, ki je dovolj zmogljiva, da upraviči svoj namen in je dejansko lahko uporabna za podjetja. Menimo, da nam je to uspelo, saj se je aplikacija dobro obnesla na testni množici podatkov. Testna množica podatkov je bila sestavljena iz petnajstih različnih odprtokodnih projektov, ki smo jih našli na GitHubu. Podatki o projektih so bili pravilno prebrani in nato predstavljeni skozi uporabniški vmesnik, ki služi kot pomoč pri odločanju o uporabi določene tehnologije ter tudi pri iskanju oseb z izkušnjami, ki bi lahko pomagale pri problemih, ki se lahko pojavijo pri uporabi določene tehnologije.

Drugi cilj je bil razviti aplikacijo, ki bo enostavno razširljiva. Dodajanje podpore novim programskim jezikom in tehnologijam je enostavno, vendar je omejeno na koncept, ki smo ga zastavili (branje datotek in iskanje izrazov znotraj njih). V primeru, da bi želeli pridobiti kompleksnejše informacije (npr. z uporabo umetne inteligence), pa bi morali arhitekturo aplikacije nekoliko spremeniti.

Menimo, da bi uporaba aplikacije pozitivno vplivala na pretok informacij med razvijalci v podjetju ter olajšala odločanje o izbiri tehnologij in knjižnic.

8.1 Nadaljnje delo

Idej za izboljšave in nadgradnje je precej, zato je najbolje, če jih razdelimo glede na komponente aplikacije in navedemo tiste, ki bi po našem mnenju prinesle največjo vrednost.

Administracijski vmesnik

- Vizualne izboljšave in boljša uporabniška izkušnja, kar bi lahko dosegli tako, da bi knjižnico ActiveAdmin bolj prilagodili, ali pa jo zamenjali z lastno rešitvijo.

Osrednji del aplikacije

- Izboljšava uporabniške izkušnje in izgleda aplikacije, še posebej na mobilnih napravah.
- Naprednejši iskalnik, ki vsebuje več možnosti filtriranja ter razvrščanje rezultatov po relevantnosti.
- Pregled vseh jezikov, tehnologij in knjižnic projektov z možnostjo naprednega filtriranja.
- Profil razvijalca s podatki o njegovih projektih in znanjih.
- Internacionalizacija uporabniškega vmesnika. Trenutno smo zaradi splošne razširjenosti uporabili angleščino, dobro pa bi bilo podpreti tudi druge jezike.

Procesiranje podatkov

- Podpora drugim programskim jezikom (Python, Java ipd.), ogrodjem in tehnologijam.
- Podpora preostalim ponudnikom gostovanja repozitorijev (npr. Bitbucket, Gitlab) in implementacija funkcionalnosti, ki bi omogočila, da posamezna aplikacija pošilja podatke naši aplikaciji.

- Trenutno dobimo podatke o knjižnicah le za primarni programski jezik v repozitoriju. Ker repozitorij po navadi vsebuje več programskih jezikov, bi bilo smiselno pridobiti knjižnice za vse jezike.
- Paketi v jeziku Ruby nimajo definiranih ključnih besed, zato iskanje po njih v tem primeru ni mogoče. To bi lahko rešili z uporabo algoritma ali pa storitve, ki zna na podlagi opisa zgenerirati ključne besede.

Spletno aplikacijo bi bilo mogoče razširiti tudi v oblačno storitev, kar pomeni, da bi ena instanca aplikacije upravljala s podatki več podjetij. V tem primeru bi morali nekoliko spremeniti podatkovni model ter zelo dobro poskrbeti za varnost podatkov.

Literatura

[1] "Web 2.0." [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Web_2.0 [Dostopano 1.6.2016].

[2] K. Purer, "PHP vs. Python vs. Ruby – the web scripting language shootout," July 2009.

[3] J. K. Ousterhout, "Scripting: Higher-level programming for the 21st century," *Computer*, vol. 31, pp. 23–30, Mar. 1998.

[4] D. Spinellis, "Java makes scripting languages irrelevant?," *IEEE Softw.*, vol. 22, pp. 70–71, May 2005.

[5] P. Jeffrey L. Duffany, "Choice of language for an introduction to programming course," in *"Excellence in Engineering To Enhance a Country's Productivity"*, Twelfth LACCEI Latin American and Caribbean Conference for Engineering and Technology (LACCEI'2014), Universidad del Turabo, Gurabo, PR, USA, 1998.

[6] G. Seshadri and G. S. Raj, *Enterprise Java Computing: Applications and Architectures*. Cambridge University Press, 1999.

[7] M. Björemo and P. Trninić, "Evaluation of web application frameworks-evaluation of web application frameworks with regards to rapid development.," 2010.

[8] "Model-pogled-nadzornik." [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Model-view-controller. [Dostopano 1.6.2016].

[9] "Kateri programski jezik ima najboljši paketni upravljalec?." [Online]. Dosegljivo: https://blog.versioneye.com/2014/01/15/which-programming-language-has-the-best-package-manager/ [Dostopano 1.6.2016].

[10] "Deset najbolj priljubljenih podatkovnih baz v letu 2016." [Online]. Dosegljivo: https://blog.jooq.org/2015/10/15/the-10-most-popular-db-engines-sql-and-nosql-in-2015/ [Dostopano 1.6.2016].

[11] "NoSQL." [Online]. Dosegljivo: https://en.wikipedia.org/wiki/NoSQL [Dostopano 1.6.2016].

[12] "HTML." [Online]. Dosegljivo: https://en.wikipedia.org/wiki/HTML [Dostopano 1.6.2016].

[13] "HTML5." [Online]. Dosegljivo: https://en.wikipedia.org/wiki/HTML5 [Dostopano 1.6.2016].

[14] "CSS." [Online]. Dosegljivo: https://en.wikipedia.org/wiki/CSS [Dostopano 1.6.2016].

[15] "Primerjava predprocesorjev CSS." [Online]. Dosegljivo: http://code.tutsplus.com/tutorials/sass-vs-less-vs-stylus-preprocessor-shootout--net-24320 [Dostopano 1.6.2016].

[16] "Ogrodja za CSS." [Online]. Dosegljivo: https://en.wikipedia.org/wiki/CSS_frameworks [Dostopano 1.6.2016].

[17] "JavaScript." [Online]. Dosegljivo:
https://en.wikipedia.org/wiki/JavaScript [Dostopano 1.6.2016].

[18] "CoffeScript." [Online]. Dosegljivo:
http://coffeescript.org/ [Dostopano 1.6.2016].

[19] "TypeScript." [Online]. Dosegljivo:
https://en.wikipedia.org/wiki/TypeScript [Dostopano 1.6.2016].

[20] "ECMAScript." [Online]. Dosegljivo:
https://en.wikipedia.org/wiki/ECMAScript [Dostopano 1.6.2016].

[21] "Kompatibilnost ECMAScript 6." [Online]. Dosegljivo:
http://kangax.github.io/compat-table/es6/ [Dostopano 1.6.2016].

[22] M. Jazayeri, "Some trends in web application development," in *Future of Software Engineering, 2007. FOSE'07*, pp. 199–213, IEEE, 2007.

[23] "Npm in upravljanje paketov na strani odjemalca." [Online]. Dosegljivo:
http:
//blog.npmjs.org/post/101775448305/npm-and-front-end-packaging
[Dostopano 1.6.2016].

[24] "Tehnologije na strani odjemalca - raziskava 2014." [Online]. Dosegljivo:
https://medium.com/@rogerdudler/
front-end-technology-stack-survey-2014-809f7a8c92f3#.193l5mqet
[Dostopano 1.6.2016].

[25] "Pisanje modularne kode JavaScript." [Online]. Dosegljivo:
https://addyosmani.com/writing-modular-js/ [Dostopano 1.6.2016].

[26] "Nadzor različic." [Online]. Dosegljivo:
https://en.wikipedia.org/wiki/Version_control [Dostopano 1.6.2016].

[27] "Git." [Online]. Dosegljivo:
https://en.wikipedia.org/wiki/Git_(software) [Dostopano 1.6.2016].

[28] "GitHub." [Online]. Dosegljivo:
https://en.wikipedia.org/wiki/GitHub [Dostopano 1.6.2016].

[29] "Upravljanje infrastrukture." [Online]. Dosegljivo:
http://devops-knowledge-base.readthedocs.io/en/latest/scm/
infrastructure_orchestration.html [Dostopano 1.6.2016].

[30] "Orodja za nameščanje." [Online]. Dosegljivo:
http://blog.takipi.com/
deployment-management-tools-chef-vs-puppet-vs-ansible-vs-saltstack-vs-fabric/
[Dostopano 1.6.2016].

[31] "Capistrano." [Online]. Dosegljivo:
https://github.com/capistrano/capistrano/ [Dostopano 1.6.2016].

[32] "Zvezna integracija v oblaku." [Online]. Dosegljivo:
https:
//strongloop.com/strongblog/node-js-travis-circle-codeship-compare/
[Dostopano 1.6.2016].

[33] M. Jurglič, "Testiranje programske opreme z uporabo ogrodja Ruby on Rails." Ljubljana: Fakulteta za računalništvo in informatiko, 2014.

[34] "Vodič za Ruby on Rails." [Online]. Dosegljivo:
http://guides.rubyonrails.org/v2.3.11/getting_started.html
[Dostopano 1.6.2016].

[35] "Arhitektura ogrodja Ruby on Rails." [Online]. Dosegljivo:
http://adrianmejia.com/blog/2011/08/11/
ruby-on-rails-architectural-design/ [Dostopano 1.6.2016].

[36] "Sidekiq." [Online]. Dosegljivo:
https://github.com/mperham/sidekiq/wiki [Dostopano 1.6.2016].

[37] "Paralelizem in sočasnost v Ruby-ju." [Online]. Dosegljivo:
https://www.toptal.com/ruby/

ruby-concurrency-and-parallelism-a-practical-primer [Dostopano 1.6.2016].

[38] J. Worsley and J. D. Drake, *Practical PostgreSQL*. "O'Reilly Media, Inc.", 2002.

[39] "Platforma kot storitev." [Online]. Dosegljivo: https://en.wikipedia.org/wiki/PaaS [Dostopano 1.6.2016].

[40] "Heroku." [Online]. Dosegljivo: https://devcenter.heroku.com/articles/how-heroku-works#defining-an-application [Dostopano 1.6.2016].

[41] "Vzorec objavi-naroči v ogrodju Rails." [Online]. Dosegljivo: https://www.toptal.com/ruby-on-rails/the-publish-subscribe-pattern-on-rails [Dostopano 1.6.2016].

[42] "JSON." [Online]. Dosegljivo: http://www.json.org/json-sl.html [Dostopano 1.6.2016].